6. Release B Cross Subsystem Design Features

The design of the ECS applications is based on a number of common software architecture principles. They ensure that (1) ECS applications will be able to interoperate, (2) ECS applications are easier to maintain, support and operate; and (3) software components can be reused across subsystems, thus reducing the implementation effort and risk.

This section elaborates on seven of these architectural concepts. They are:

- Distributed Communications Architecture,
- Security Architecture,
- External Interface Architecture,
- Systems Management Architecture,
- User Interface Architecture,
- Earth Science Query Language (ESQL) Architecture, and
- Mode Management Architecture

Within the ECS architecture, responsibility for design and development components is allocated in an unambiguous manner. Aspects of these designs, therefore, will appear in the various detailed design documents of the subsystems which have the responsibility for their implementation. However, the concepts are presented in this overview because they cross subsystem boundaries and their interconnections and rationales are lost when viewed solely within the context of a specific ECS subsystem.

6.1 Distributed Communications Architecture

6.1.1 Overview

The primary goal of the distributed communication architecture is to provide a software infrastructure for current and future ECS development. This infrastructure must meet the science and technology requirements of Release B, and provide a clear evolutionary path to the system capabilities, technologies, and capacities required for Release C and beyond. In order to achieve this goal, the Release B design was influenced by a number of design drivers. Foremost among them are the following:

- Extensibility - During the lifetime of ECS the system must accommodate growth in the areas of value added providers, new data types and services, and interoperability with other information systems. The distributed object architecture provides a uniform mechanism for referencing and accessing ECS data, objects, and services.
- Evolvability - Due to the rapid growth in processing hardware, storage devices, network communications, and information services the ECS must support a continuous evolution of it’s components.

- Scalability - The ECS design must scale easily to accommodate peak processing loads or sustained loads beyond nominal system loads defined in the technical baseline for Release B.

The distributed communications architecture provides the framework for meeting the needs of the design drivers. Following the client/server architectural model, the distributed communications architecture allows client and server applications to cooperate without direct knowledge of each other. Service requests are made by invoking methods of a local "proxy" object within the client's address space. The proxy object, in turn, communicates with the server application to perform the service request. This communication may occur across address spaces and platforms, and is transparent to the application developer and the user.

This mechanism provides a layer of isolation between the client application and the server implementation such that modifications to the server implementation can proceed without changing the client implementation, so long as legacy methods provided by the server continue to be supported. This allows ECS to add new servers, upgrade processing or storage hardware, and to add new services without requiring modifications to clients of the server. ECS will use DCE, encapsulated via the Object-Oriented DCE (OODCE) COTS product for Release B. The architecture allows for a managed transition to other technologies (e.g., CORBA) in subsequent releases. The Release B distributed communications architecture is comprised of the following key components:

- Distributed Object Framework (DOF)
- Universal References (URs)
- Process Framework(PF)
- Server Request Framework

Each of these topics will be discussed in the following sections.

6.1.2 Distributed Object Framework

Object-Oriented applications are assembled from a number of interrelated objects. Each object is characterized by a set of attributes and methods. Each object has a clear interface that identifies the methods a user can invoke. The object that requests information is called the requester and the object that provides a service is called the provider. Each provider object takes requests for operations that it has identified in the interface, performs the computations, and passes the results back to the requester. Object-oriented application development consists of defining and instantiating the objects and passing messages (invoking methods) between the objects to achieve its objective.

In single address space applications, all objects reside in the same address space. In the distributed object framework, objects are distributed in multiple address spaces, spanning heterogeneous platforms. Objects can reside anywhere in the network, but the basic contract between a provider object and requester object is the interface that the provider advertises for a requester to use. Objects can be spread across the network based on efficiency, availability of data, requirements for autonomy, etc. From the perspective of the requester of a service, the object location (location independence) and invocation (invocation independence) should be the same no matter where the object physically resides. Invoking methods amounts to passing messages between objects. If the

objects are in different address spaces, then the messaging is done via a network. The client/server paradigm supports this kind of communication. In this paradigm, one side of the session (client) is allowed to make requests, while the other side (server) may only make replies. Remote Procedure Call (RPC)is one communication method that is used to implement the client/server paradigm.

The distributed object framework provides the underlying infrastructure for remote object creation and method invocation. In addition, it also provides a set of core services with distinct functionality to make the development of the distributed applications easier. The core services are naming, security, threads, time and rpc. DOF interacts with the Naming Service to save and retrieve service locations. Similarly it interacts with the Security Service to ensure security. In addition to core services, CSS will be providing DFS which allows users and applications to easily access files across a distributed system.

In order to understand the distributed communication supported by DOF it will be illustrative to review a functional framework provided by standard DCE. In this model, the server generally provides certain operations (which are typically termed subroutines/functions) for the clients to use. In that sense, a normal program can be broken down into a number of subroutines and servers which implement the subroutines. Clients call these subroutines as if they are local. When a client invokes one of these remotely implemented functions, program execution transfers from the client to the server where it is processed. Once the execution is done, the result is passed back to the caller. The client and the program execution flow will then be turned back to the client. In order to achieve the above client/server interaction, a standard interface definition language (IDL) is used to express the interface in a clear way. This is a pseudo language for which mappings exist so that the interface expressed in IDL can be converted to high level languages like C and C++ using the IDL compiler. Once the interface is expressed in a standard language, any requester who wants to make an invocation can do so by adhering to the signatures present in the interface. The IDL typically supports standard types, obeys some lexical rules and has a language syntax to express the interface in a crisp and unambiguous way by preserving the semantics of the interface.

Since the data formats or internal representation of data may be different on different platforms, the RPC mechanism provides a way to convert the data into a standard format so that both the receiver and the sender would interpret the data in the same way. The process of converting the data into a standard format is called marshaling and the process of converting the data from the standard format into a platform's internal format is called unmarshalling. Thus, this paradigm deals at the program function level and has no notion of objects.

An object framework is somewhat like the functional framework just described, but instead of differentiating at a function level, it differentiates at an object level. The object's behavior is captured in the interface definition language. The object's implementation is carried on remote hosts, which are responsible to execute procedures, update the object state and return the results. This paradigm makes use of inheritance while defining new interfaces by inheriting existing interfaces. Implementation inheritance may also be possible as long as the implementation of the super class exists within the scope of the current implementation. This paradigm also needs a standard interface definition language and provides the mechanism to marshal and unmarshal standard types. In this paradigm, an executing program (client) can instantiate an object at any host that provides the implementation of that object and query that object to do certain operations. In order to do that, two objects are created: one at the client and one at the server. The object created at the server is the real object that implements the behavior of the object. The object created at the

client is called a surrogate object, whose main purpose is to marshal/unmarshal the arguments, make call to the real object, and get the results back to the calling program. From the client's perspective, the call is carried locally. The surrogate object does all the underlying work: locating the server that is offering the service, binding to it, setting security preferences, instantiating the server object if necessary through the use of life cycle services, and finally invoking the method. This is transparent to the client and is done through the use of IDL and the supporting framework.

In DCE, the interfaces are defined using the DCE Interface Definition Language and are processed by a compiler called IDL, which generates system data structures and communication stubs for which generates system data structures and communication stubs for the client and server. OODCE uses the same IDL language, but an enhanced version of the compiler is used to process the interface specification. The compiler program is called idl++. The idl++ compiler generates the client and server stub and header files needed for a DCE RPC interface. idl++ also generates a number of C++ files that provide communication between OODCE clients and servers. Even though IDL++ generates C++ language stubs, it does not support interface inheritance and class attributes.

Application programmers implementing the client server application need to develop three parts: an application client part which invokes the service, an application server part that implements the service, and an application server main (driver) part which actually creates and runs the application server as a separate process and provides all the functionality needed at the server side. The application program carries on all the interaction with the underlying core services like naming, security, threads, and time through the DOF for normal operations. Interfaces to these underlying core services are also provided.

More detailed information on the Distributed Object Framework can be found in Section 4.3 of the Release B CSMS Communications Subsystem Design Specification, 305-CD-028-002.

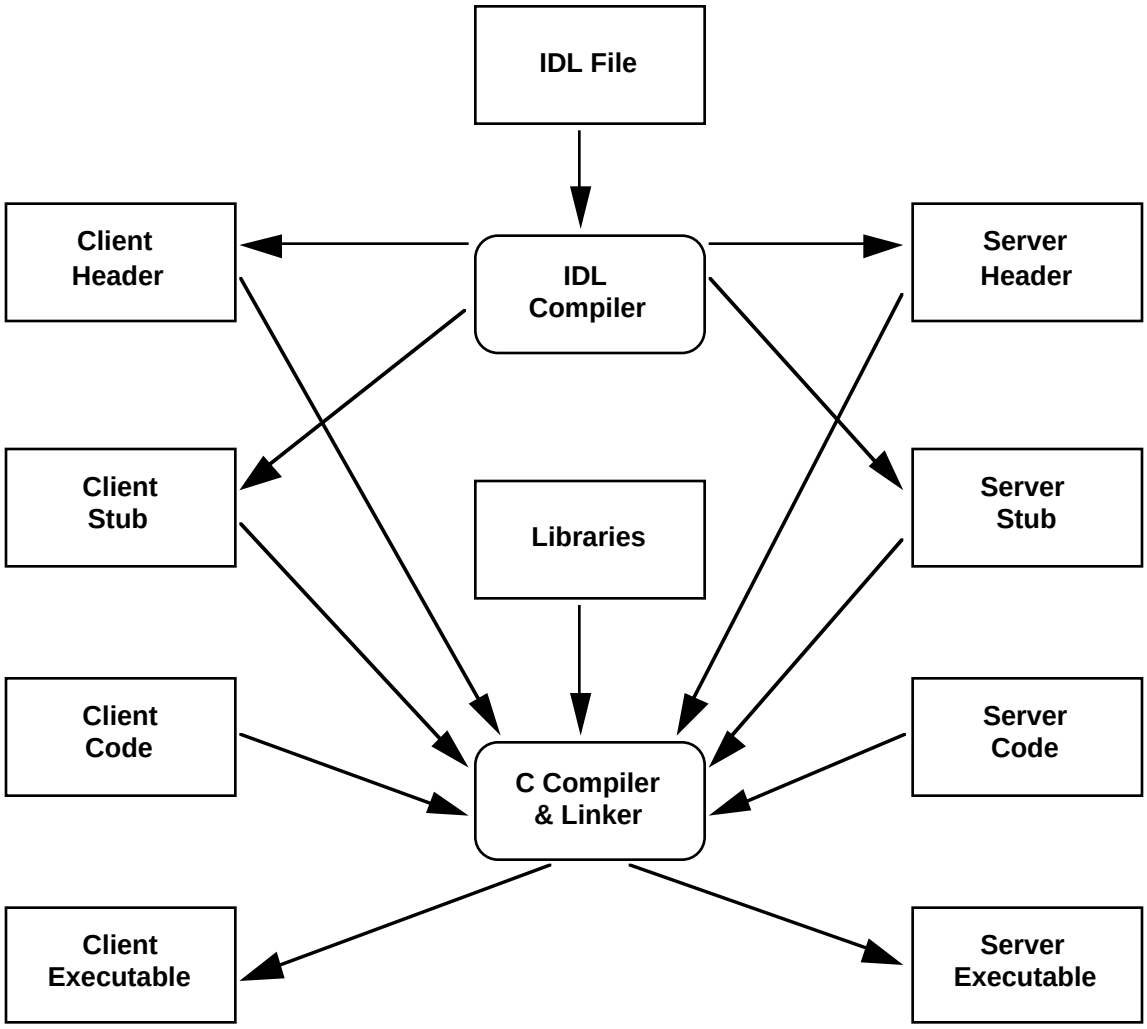

Figure 6.1.2-1 DOF Client/Server Application Development Process

6.1.3 Universal References (URs)

Universal References (URs) provide applications and users a system wide mechanism for referencing ECS data and service objects. Once a UR is made for an object, the object can be disposed of and later reconstituted from the UR. URs can refer to objects that may be local to an address space, or remote.

URs are implemented as a framework which provides objects the capability to create URs for themselves, distribute URs throughout the system, then use these URs to reconstitute and/or access the original object. URs themselves are objects which can externalize themselves into an ASCII representation, then internalize from their ASCII representation back to objects. The content of the externalized UR is human transcribable(ASCII representation) and transport friendly. While the UR mechanism guarantees reliable data externalization and internalization, the content of each

type of UR is application specific. Only the object that initially provides the UR (from now on we will call this the "UR Provider") is allowed to access and understand its content. URs are strongly typed to enforce appropriate access control to internal data both at compile time and during runtime. Since URs are typed and have object specific data in them, separate UR object classes exist for each UR Provider object class referred to. All of these UR classes use the mechanisms provided by the UR framework. The framework also provides UR Providers support for their common requirements. The following diagram, Figure 6.1.3-1, illustrates the functions of the various frameworks.

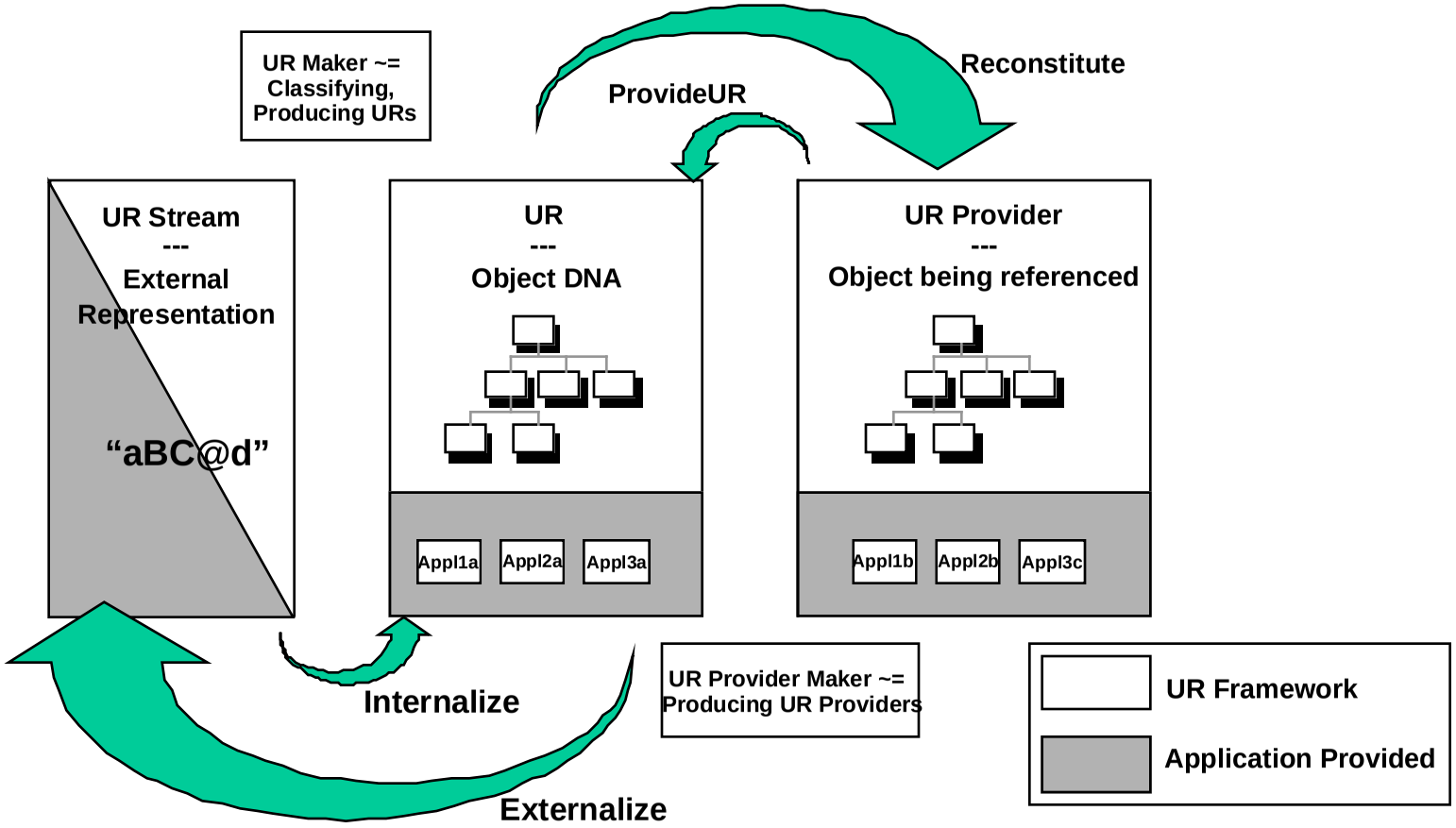

Figure 6.1.3-1. Functions of Different UR Frameworks

From left to right, going from top to bottom,

- the UR Provider framework supports common functions for all object referenced by URs. This framework allows URs for the UR Provider to be extracted and later reconstituted.
- ECS application objects subclass this framework to enable themselves to be referred to by URs.
- the UR framework supports common functions for referring to UR Providers; this function will allow URs to be externalized and internalized.
- ECS application UR objects subclass this framework and will hold the application specific state data to uniquely identify the application object referred to.
- URs can be externalized to ASCII strings that are managed by the user.

The object paradigm uses techniques of abstraction and inheritance to create designs that are resilient to change. ECS will extend the data and services it provides. The UR and UR Provider Frameworks are designed to support both abstraction and inheritance. An abstract UR that provides a defined set of data and services can be passed to a UR consumer. The consumer is not required to know the actual concrete UR object they are working with. When these abstract URs are used to access the object they refer to, the concrete object is accessed. However, the consumer is not required to know which concrete object they are using. They are only required to know the interfaces to the abstract object. This capability of working with abstract URs and abstract objects allows ECS to extend, through inheritance, the data and services it provides with minimal changes to applications. The framework supports the dynamic introduction of new remote objects that meet a given pre-existing abstract interface.

A corollary design component of URs concerns backward compatibility. Objects providing URs will change over time. The URs they provide as references to themselves will then also probably change. The UR and UR Provider Frameworks allow a UR Provider to support multiple UR types as long as a backward compatibility feature exists in the provider. This design component will mitigate problems of end users and applications using URs dispensed from prior software versions.

As described, URs have an extensible and evolvable design which allows the UR types to grow and change over time. A very important specialization of URs is used in the SRF design, where URs are tailored to support location of services. The UR framework is independent of the communication infrastructure and does not couple the ECS architecture to OODCE. It provides a general mechanism for keeping references to logical entities for long periods of time, without the computational costs of maintaining those objects in memory.

6.1.4 Process Framework(PF)

The ECS contains several infrastructure features which facilitate the implementation of client-server applications. The Process Framework provides an extensible mechanism for ECS Client and Server applications to transparently include these infrastructure features. Therefore, its utility grows with future releases of ECS. Furthermore, the framework is used solely by ECS custom developed applications and as such is not meant for COTS applications. The primary objective of the PF is to ensure reusability for all ECS Client and Server applications. This is achieved by encapsulating the implementation details of ECS infrastructure services and removing the need for programmers to rewrite common initialization code.

In general, the following capabilities are needed in the ECS client and server applications and have to be accommodated in an appropriate fashion:

(1) Ability to initialize the process application and infrastructure in a consistent way and provide some basic process information

(2) Interface to Mode Management

(3) Interface to Event-&-Error logging

(4) Ability to set Naming/Directory Service options

(5) Ability to set Security management parameters

(6) Support for Life Cycle services

(7) Interface to Asynchronous Message Passing

(8) Interface to Server Request Framework, and

(9) Interface to the Batch FTP service

A two step approach is used to develop the process framework. First, a process classification for the ECS project is developed from the client/server perspective. Then, the required capabilities are allocated at different levels of abstraction for each process type. The details of the above steps are presented below.

Process Classification

Figure 6.1.4.2-1 presents classification of ECS application processes. A generic process can be specialized in client process and in server process. The former can be specialized into a gateway client (client external to the ECS system and connected to it through a gateway) and DCE client (client in the ECS system which uses DCE communication mechanisms). A server process can be specialized into an unmanaged server process and a managed server process.

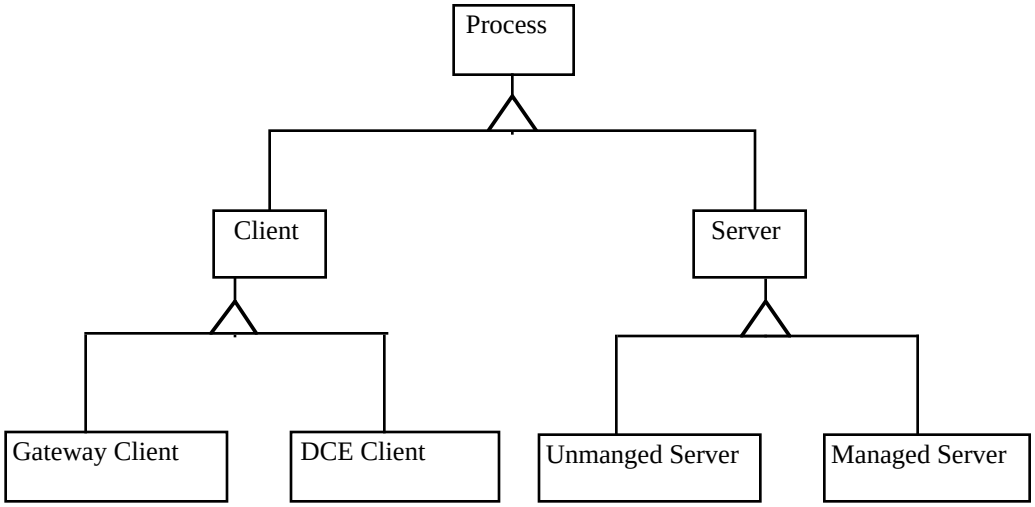

Figure 6.1.4.2-1. ECS Process Classification

In this context, a Managed Server Process is a process controlled by the management agent services provided by MSS. An Unmanaged server process is a process created by another process for its specific use and is not under the direct control of the management agents. The name must not confuse the reader. An unmanaged process is managed as well, but it is managed by its parent process instead of MSS. Finally, it should be noted that a process, which is both client and server, is considered a server.

Capability Allocation

All the capabilities that are common to all the processes need to be incorporated at the generic process level in the hierarchy described above. These include, the basic process information capabilities, and interfaces to Mode Management and Event Logging. The ability to locate services and set security preferences is provided to the client processes and server process in different ways. Therefore, this functionality is implemented differently in the client and server but the interface is common.

General services such as asynchronous messaging, Server Request Framework access, and batch FTP that are needed by all server processes are provided by interfacing with the Message Passing Service, the Server Request Framework and the batch FTP facility respectively. Full Life cycle functionality is provided to every Server process. It has to be tailored in a different way for Managed or Un-Managed Servers. The former has to be implemented with the management instrumentation classes being developed by the Management Subsystem (MSS). The latter has to be implemented by the parent Server which spawned the un-managed server.

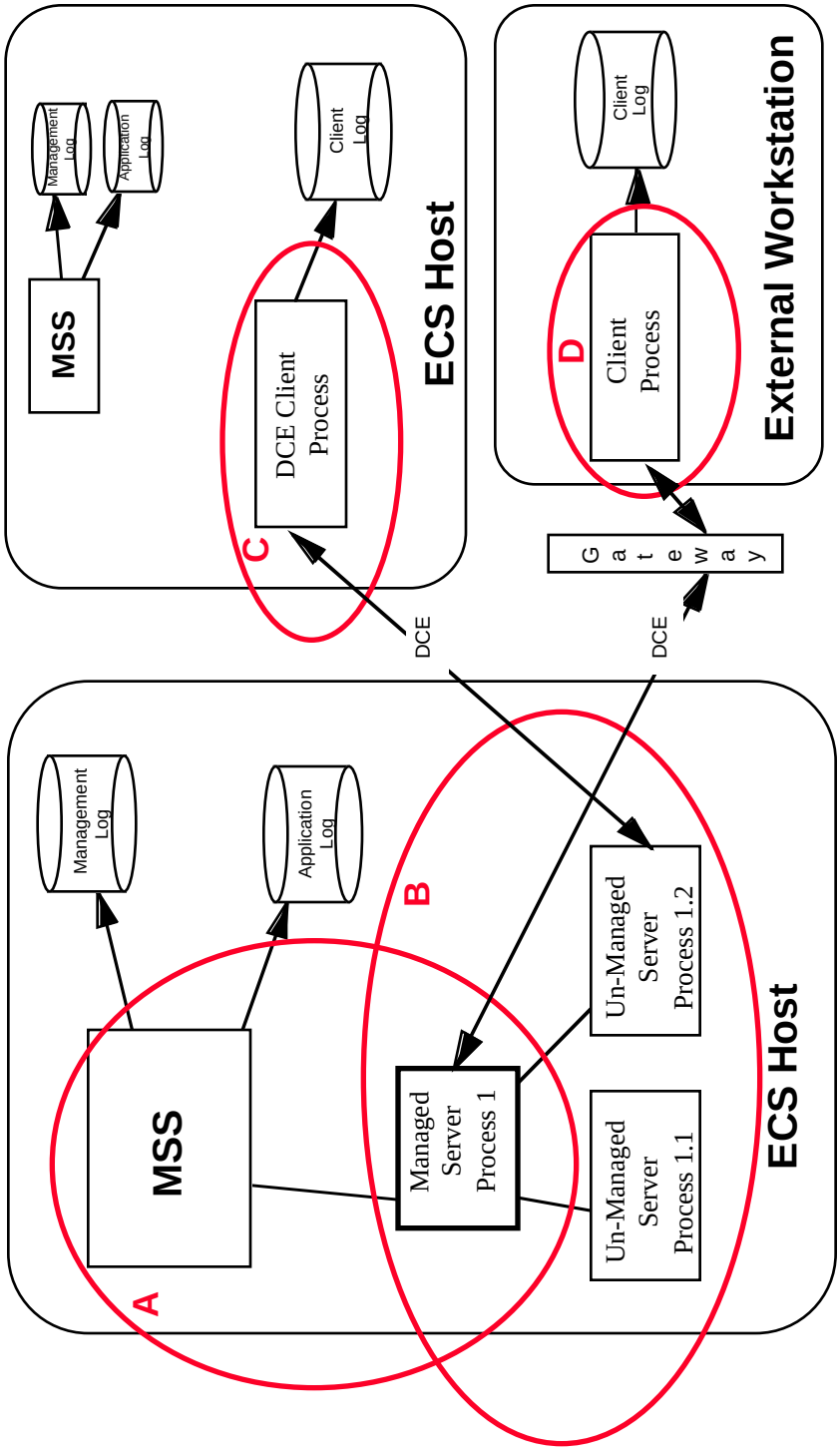

Figure 6.1.4.2-2. Process Framework Context

Process Framework Context

Figure 6.1.4.2-2 identifies the context in which the above described processes operate. The ellipse A, B, C, and D locate the contexts for Managed Server Processes, Unmanaged Server Processes, DCE Client Processes, and Gateway Client Processes respectively. A Managed Server process is connected to the MSS sub-system (ellipse A). It can also spawn other processes. These latter processes might be other Managed Server Processes as well as Unmanaged Server Processes. Unmanaged Server Processes are connected to the parent process which has spawned them (ellipse B). They are not directly connected to MSS, and MSS is not aware of their existence. Their world is limited to the application domain. They are called unmanaged because MSS doesn't manage them directly. Client Processes have less constraints. They are simply clients of ECS server applications with an external interface to log information in the *Client Log*. Only the DCE clients (ellipse C) are part of the ECS and for only these kind of client processes is the framework applicable.

6.1.5 Server Request Framework

The Server Request Framework provides a system-wide mechanism for constructing ECS Servers and Client/Server APIs. In general, it provides a single common implementation of asynchronous request processing services. ECS Client applications use SRF to handle callbacks associated with asynchronous requests and notifications associated with subscriptions. The Server uses SRF to track service requests, accept callbacks (e.g., event notifications) and sends them to the corresponding client object. This infrastructure provides factories for creating server objects (and client counterparts) dynamically. It will create and track service requests, accept call backs (e.g., event notifications) and sends them to the corresponding client object. It automatically tracks objects which can handle incoming requests or accepts callbacks and dispatches the requests or callbacks to the correct object, and matches up synchronous requests with their responses. Client applications can send requests to server objects identified by their UR - the infrastructure will resolve the UR to the correct server object. Finally, the infrastructure provides factories for creating server objects (and the client counter parts) dynamically. The Server Request Framework is integrated with the Process Framework.

SRF will be used by ECS clients to access ECS services. It is intended to be supported by all servers that handle requests and thus by all clients that interact with those servers. It will not be used by MSS or other subsystems that do not need the request queuing and asynchronous features of SRF.

SRF is structured into two layers. These two layers together provide the SRF functionality. The SRF Client/Server layer is used directly by applications. The SRF Message Handling Layer is used internally by SRF to enhance the underlying CSS Messaging Layer. However, server developers will need to use some of the SRF Message Layer classes directly for initiating communication.

The following figure shows the software layers in a typical SRF based application. The SRF Framework provides two layers of the application: The client/server layer and the Message Handling layer. These layers build on the message passing (implementation one) layer provided by the CSS subsystem.

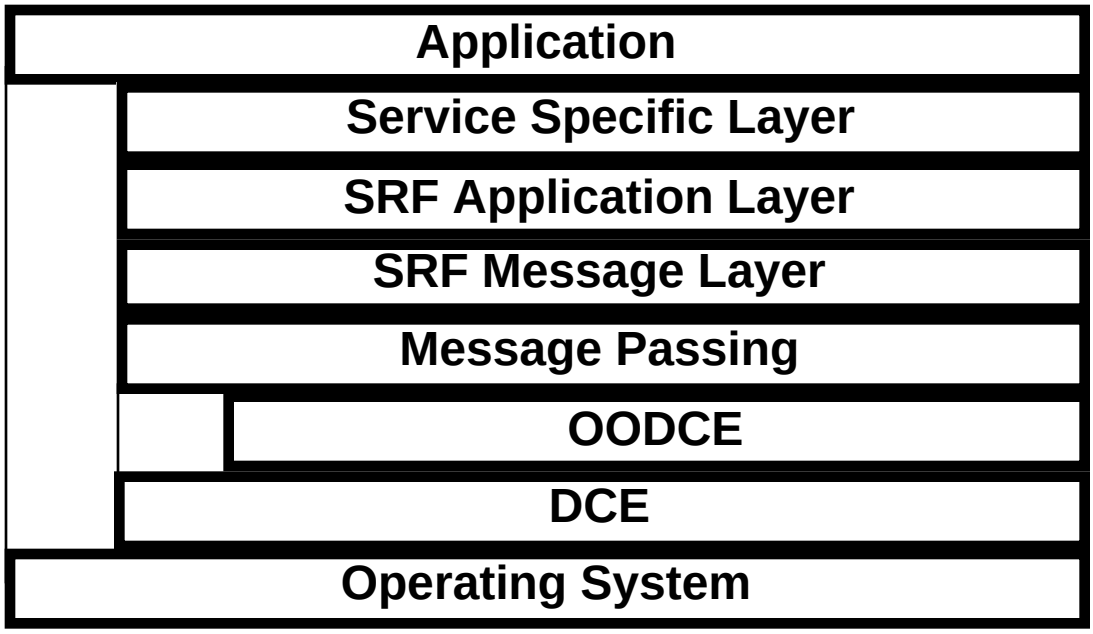

Figure 6.1.5-1. SRF Layering

The SRF Client/Server layer provides a set of 5 generic object classes for both clients and servers. The figure below presents a high-level overview of the framework.

The infrastructure provides

- generic asynchronous request object classes for the client and server side
- object classes which implement the communication between client and server and dispatch requests or callbacks to the correct client or server object
- a generic message class for transmitting requests, responses, and call backs

There will be two client side implementations of the infrastructure: one for clients which operate in an X-windows environment; and one for other types of clients. The special version for X-Windows is needed because X-windows and Motif do not support multi-threading. The infrastructure will insulate the X environment from the threads which are necessary to support asynchronous communication.

6.1.6 Request Tracking

The Request Tracking Key Mechanism is intended for use by the developers of ECS applications to report request status changes back to a central database to be displayed to an operator. The mechanism is also used to report resource cost that was collected during the life of the request. The cost data is reported back and used for cost accounting. The types of requests which are tracked are as follows: Product Orders, Invest Requests, User Requests, and Operator Requests. The mechanism also accounts for spawning of requests. When an ECS applications creates one or more sub-requests for a request, a parent-child type of relationship is established in the mechanism so that the spawned requests can be tracked independently of each other and the operator will be able

to get status information for the entire tree of requests. For more detailed information, see the Accountability Management section of 305-CD-029-002, Release B CSMS System Management Subsystem Design Specification.

6.2 Security Architecture

6.2.1 Driving Requirements

ECS is part of an international science and research program accessible to users on a global basis. Built on an open-computing network architecture to facilitate access by a wide variety of users, the driving requirements of its security architecture are data integrity, availability, and confidentiality. The project vision is open access to well-pedigreed data in which users may have absolute faith, while maintaining certain proprietary and administrative data in confidence.

Because of the scientific nature of the project, integrity of project information is paramount. Scientific formulae, streams of collected data, algorithms and modules used to process or measure, data files, archives, historical records control instructions and telemetry data all must be stored with absolute confidence in the integrity of the stored material. In addition, information to support accounting and billing for ECS products and services; as well as requests for on-demand data acquisition and processing, must be protected against unauthorized access. Users expect the system to guard against tampering with the source materials of the archive not only during storage, but during product generation, and distribution. Integrity threats are manifested through unauthorized access or use, leading to change, alteration or modification of information resources. The security architecture must provide adequate safeguards against these threats.

The ability to have assured use of project resources is vital to instill confidence in its users. Indeed, the users of ECS can be characterized as investors. Whether they are data providers (investing the fruits of their labor in the vision of an open, accessible archive) or data consumers (investing tax dollars, grant dollars, or corporate investment dollars), they have a right to expect availability. Availability is also important for more mundane, but just as vital reasons, for example, to satisfy timing and coordination requirements for scientific study and research experiments. Availability needs translate into two categories: fault tolerant features to preclude failure or operational disruption; and recovery actions, to enable timely resumption of operational activities and minimize the length of the disruption. Availability threats are manifested through denial of use actions, either physical in nature (e.g., explosion, flooding) or logical (e.g., computer virus or other intrusive software, or flooding, i.e., swamping a system component such as a gateway with such a large amount of messages or requests that other users find it difficult to get serviced). Because of the increasing complexity and interdependencies in large integrated systems, a failure in one component could have widespread physical or logical impact.

Confidentiality requirements exist because some of the information within ECS requires special protection. This includes operational or control information that is time-critical or used to control mission operations, specific user product requests and account information, and information of a private nature on individual users on the system. Information of this nature, if compromised, could result in damage or harm to ECS or to individuals. In addition, in some few instances archived data may be held as proprietary for a period of time, before it is made generally accessible. This may be for a short interval, while the data producer is exercising quality control testing over a new

product, or of a longer duration, for example, provided to a data producer as an inducement to participate in the ECS system.

6.2.2 Information Security Strategy

Physical threats are countered by physical security measures that are, generally, implemented by physical barriers or operational procedures. The subject of this Security Architecture section of the design specification is logical security, i.e., measures taken by the computer system to protect itself against threats. Within the context of the International Standards Organization (ISO) Open Systems Interconnect (OSI) model, ECS security functions are applied at the Application Layer (layers 5 and above), supported by an infrastructure (layers 4 and below) providing network, transport, and interoperability services.

Security management services are provided enterprise-wide through the Management Services Subsystem (MSS). MSS applications provide services such as fault, performance, security, and accountability management. MSS manages both ECS's network resources, but also ECS's host and application resources. It also provides administrative support to the defensive infrastructure employed (e.g., DCE, application-layer gateways, network firewalls) by ECS to defend against a variety of logical threats. MSS allocates these management services to both the system-wide and local levels. Thus, with few exceptions, management service are fully decentralized, with no single point of failure that would preclude authorized user access, nor permit unauthorized user access.

The main threats to the ECS and their countermeasures are listed in Table 6.2-1. The table also indicates the ECS subsystem which is responsible for the implementation of the countermeasures. In general, system security measures are assigned to CSMS, since logical security attacks today usually exploit the network links that each system has with the outside. Moreover, by securing the communications within the system, it is often possible to isolate the remainder of the system from a successful attack on a particular system component.

However, communications components lack the application context that is sometimes needed to provide special protection. For example, CSMS can protect the access to a service which manages private data and thus relieve that service from having to implement extensive security measures itself. However, CSMS protection is incomplete, since CSMS will know only the destination but not the specific nature of an access request. In such cases, applications will provide additional access controls; these will typically rely on the user authentication performed by CSMS, but apply application-layer rules to determine authorization. They may also depend on CSMS ACL management as a repository for persistent, global, or remotely managed authorization rules.

6.2.3 OSF/DCE Architecture

The core of information security within ECS will be based on the Open Software Foundation's Distributed Computing Environment (OSF/DCE). This technology is generally used in ECS to provide enterprise coordination among mixed computing platforms. Applications are layered atop a framework of DCE "middleware." In the present context the DCE middleware provides extra services in the area of data integrity, availability, and confidentiality. In particular, DCE provides user authentication, authorization down to the individual transaction level (for individuals or groups), and secure interoperability among applications.

Object-Oriented DCE (OODCE) provides a linkage between the object-oriented design approach selected by ECS and the DCE libraries provided by OSF. It may be viewed as an abstraction layer between the object-oriented application and the DCE middleware, making all of the DCE services (including naming, time, distributed file access, and threads, in addition to security) easily accessible to the application programmer.

Table 6.2-1. Threat/Countermeasures

Threat	Countermeasures	CSS	ISS	MSS	Other
Unauthorized use, privilege abuse	Authentication/Authorization	X		X	DM, DSS
Unauthorized use of access controlled resources	Access Control	X		X	DM, DSS
Data tampering	Data Integrity	X	X		
Electronic 'eavesdropping'	Data Privacy	X			
Unauthorized use, cracker/hacker activity	Security Audits	X		X	
Unauthorized use, abuse, virus detection, denial of service, 'cracker' and hacker detection	Intrusion Detection	X		X	
Lack of secure environment due to non-compliance of security procedures	Compliance Management			X	
Crackers and hackers, viruses, broadcast storms, unauthorized use of resources	Routing Control Firewalls Address filtering		X		
Unauthorized access to private or proprietary data within a data server	Authentication/Authorization DBMS Security	X			DM, DSS
Unauthorized alteration of processing schedules and plans	UNIX Access Controls Authentication/Authorization DBMS Security	X			PLS & DPS

DCE provides security features by employing a variation of the Kerberos model (version 5 or later) and POSIX 1003.6 Access Control lists (ACLs). Kerberos provides a strong protocol for authenticating users (proving a user is who he/she says he/she is), and thereafter provides trusted third-party services to applications, vouching to them for a previously authenticated user's identity. Secure authorization (verifying a user is authorized to perform the requested function, or access the selected data) is provided by authenticated remote procedure calls (RPCs), a form of function entitlement, and by application-embedded rules or application-invoked ACL managers, which verify the authorization of an authenticated user to any requested resource. The combination of authentication, authorization, and secure interoperability is the key security characteristic of the ECS DCE/OODCE architecture.

Figure 6.2-1 illustrates the main concepts of the ECS security architecture. The DCE Cell Configuration trade (see 540-TP-001-001, Technical Paper: Communications and Systems Management Segment (CSMS) Preliminary Design Review Trade Studies for the ECS Project) identified multiple cell as a preferred design solution with regard to scalability and evolvability concerns. A cell around each DAAC set of resources is employed to prevent general public access into production environments. In addition, each DAAC requires at least one host with a low

security, public access (a “gateway”) to encourage the use of ECS. One solution is the creation of an integration cell that 'connects' all public access points at each DAAC together and isolates the rest of the system from the security threats they pose.

The multi-cell security architecture for Release B represents the optimum middleware infrastructure and associated security, while living within the constraints of the currently available technology.

The Release B evolution path is to provide multiple cells, one-per-DAAC, providing local autonomy and improved reliability and availability across the entire (multi-DAAC) system. The cell partitioning strategy does not prohibit direct high-speed access to ECS production LANs, and provides efficient location of system resources through distributed directory and security services in every cell. By delegating the assignment of user privileges independently to each DAAC cell, privileged user access may follow site policy, with little likelihood of direct intrusion or potential for security breach by the general public. By establishing a trust relationship among the various ECS cells, distributing the privilege assignments provides autonomy while enabling users to enter ECS anywhere and have the same view of the system.

6.2.4 Security Implementation within ECS

Figure 6.2-2 illustrates how DCE security is used within ECS to enable a secure distributed computing environment. After the administrator has created an account for a user, the user can participate in a secure DCE system. Typically a user logs in at the beginning of a session through the CSS login facility server. The login facility server sends a request for authentication credentials to Authentication Server. The Authentication Server sends back the authentication credentials, called a Ticket. The Authentication Server's replay is encrypted using the user's password, so if the user can supply the right password, the replay can be decrypted and the *Ticket* can be accessed. Tickets are used by clients to authenticate themselves to servers; that is, to prove that clients are really who they say they are.

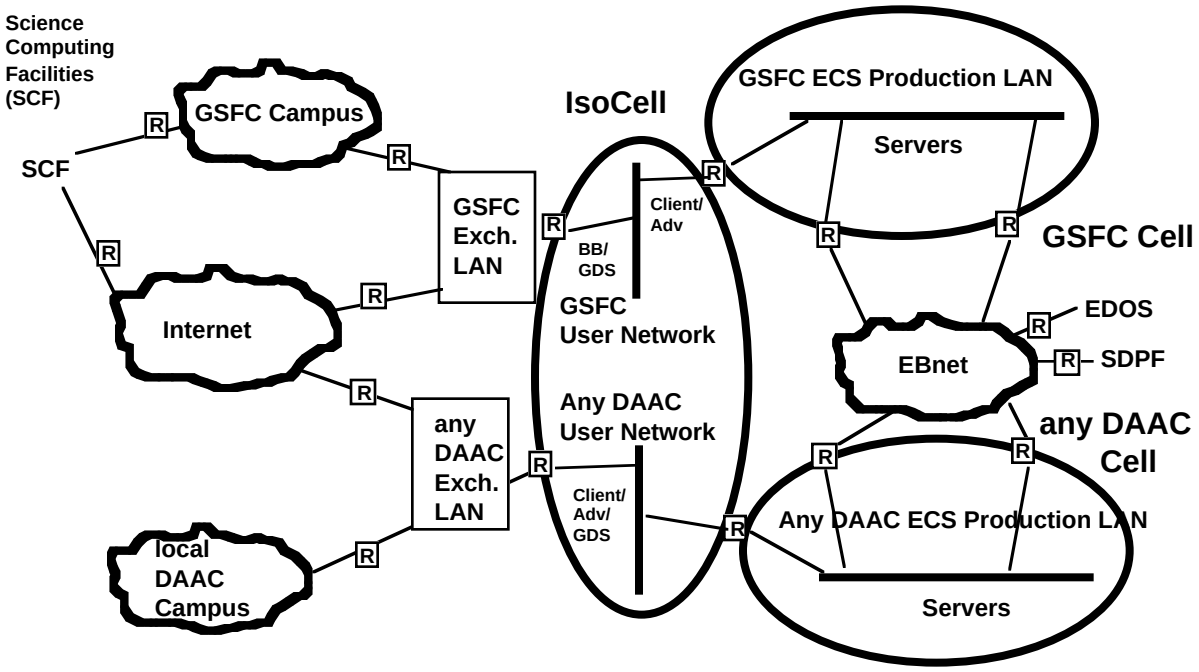

Figure 6.2-1. Release B Cells/Security Architecture

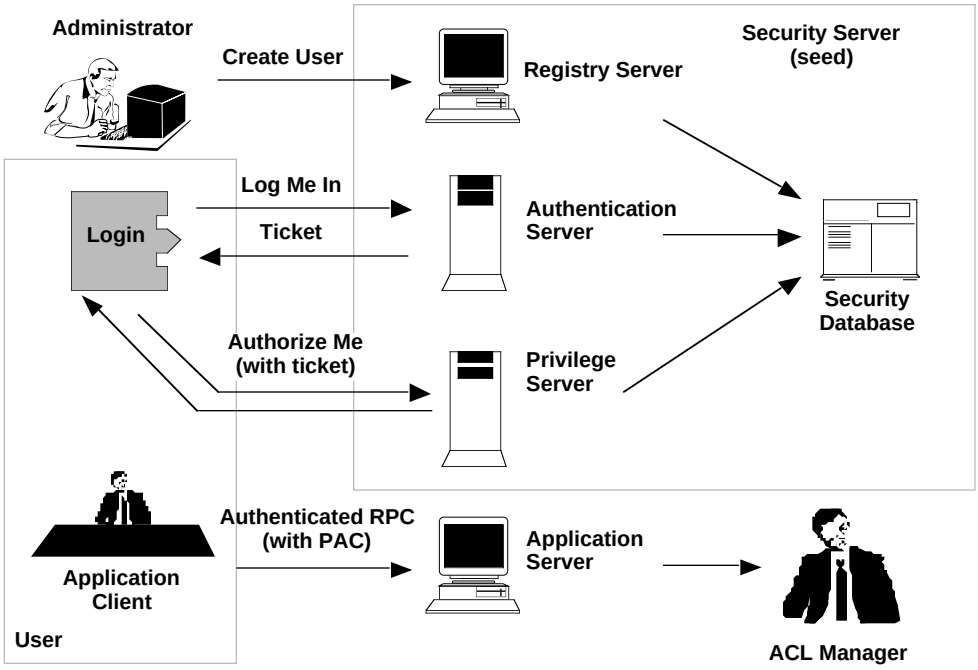

Figure 6.2-2. Use of DCE within ECS

Next, the Login Facility sends the Ticket to the Privilege Server. The Privilege Server returns authentication credentials, called a PAC (Privilege Attribute Certificate). The PAC contains authentication information specific to the user, such as which groups the user belongs to. PACs are used to authorize users; that is, to help a server decide whether users should be granted access to resources that the server manages. When the Login Facility has finished running, the user has a security environment and can communicate in a secure way with application servers. Major aspects of security include data privacy and integrity, authentication, and authorization.

Data integrity ensures that data in transit or in storage is not modified. DCE provides an option (specified by the application programmer when designing the respective interface) to add encrypted checksums to the data. DCE can also ensure that data in transit cannot be read any unauthorized parties by encrypting the data stream itself. The DBMS, UNIX file system, and access to either (protected by DCE authentication/authorization) are responsible for the integrity of data in storage.

The protection level for data privacy or confidentiality is, like data integrity, again partially determined by the application designer. Full encryption, used to guarantee the privacy of data either in transit or in storage, is expensive in terms of system performance and should be used sparingly. The same controls providing for data integrity (DBMS, file system, authentication, authorization) also are used to provide for data privacy for data in storage.

DCE also provides strong authentication, being able to verify the identity of a principal without passing his or her password in the clear. With DCE, authentication is done through a trusted third party. Authentication can be performed manually through the principal, or automatically within normal client/server authentication processes. A principal authentication, done by a user during login, is an interactive process in which the user password is supplied on the users command line, but not passed across the network to the server. A non-interactive principal authentication occurs when a host is booted and a server needs to acquire a different identity other than the default (root).

The process of checking the privileges of a principal (i.e., whether he or she is allowed to access a system resource on the network) is called Authorization. Authorization can be based on the name of the principal (this is called name-based authorization), or on the group to which the principal belongs (this is called PAC based authorization). The specific authorization type is specified upon initial session login by the selection of the appropriate authorization protocol. Group based authorization is useful when many users share similar privileges; by assigning their privileges to a group, the administration of the privileges is simplified, but it is still possible to maintain the user's identity within the system. Name based security, on the other hand, is needed when individual users have specific privileges not shared by others (as may be the case, for example, with DAAC operations personnel).

DCE manages authorizations using Access Control Lists (ACL). A component of DCE, the ACL manager, provides functionality needed for the authorization process. It defines access control permissions, creates and associates ACLs to objects, creates and manages ACL databases, and supports standard interfaces for external system. CSS provides an API via which ECS applications can inquire whether a user is authorized to obtain the service he or she requested. This interface hides the nature and the mechanics of the authorization process from the application - the application need not even "know" the user's login name or group account. ACLs will be enforced for all ECS end-users by the ECS subsystem that they connect to (e.g., DM or DSS). This puts the

burden of ACL checking on the "boundary" subsystem. For non-DCE end-user clients, either they will be represented by DCE proxy clients (e.g., gateways), or will be treated for authorization purposes as belonging to the ACL group "Guest". A corollary to this approach is that, in general only trusted application services (e.g., DM and DSS) can access DBMSs, not end-users or their client applications directly. There must be exceptions(e.g., for DBA functions), but these are restricted to a special subset of (DAAC Operations) users.

The secure web server will give ECS users and non-ECS users the ability to retrieve information using the world wide web without compromising the security of the ECS system. The web server will have the capability of supporting DCE and non-DCE based users. The web server will authenticate all DCE environment users and will authorize their access to the ECS services. Users accessing the server, via a strong authentication-capable browser will have access to restricted as well as non-restricted data and services that are available to them. These services are provided to these users based on their access privileges. Users accessing the server, via a generic browser will be able to access any non-restricted data items and services within the ECS system. This gives these users the ability to have access to some ECS data while using his preferred browser. These users may include college and grade level students, scientist from all over the world and anyone on Internet wanting to know what ECS is all about.

6.2.5 Non-DCE Based Security

Within ECS Release B, not all communications are enabled by DCE. For example, bulk data transfers regularly take place using file transfer (ftp). ECS secures these interfaces by making them an integral part of a client-server sessions which is initially established using DCE. For example, in order to initiate a file transfer, a client such as the processing subsystem needs to issue an authorized data staging request to the data server subsystem. That is, the initiating event will have had to pass DCE authentication. In addition, depending on the sensitivity of the data transfer, a Kerberized version of ftp (kftp) will be supported. Kftp transactions will use the end-user's DCE userid and password to reduce user burden. The situation is more complex at the external interfaces which ECS has with other systems or legacy software which is reused for Release B. This is further discussed in the next section of this overview, as part of the External Interface Architecture.

6.3 External Interface Architecture

6.3.1 Overview

ECS provides interfaces with external systems (e.g., TSDIS, EDOS) or legacy software (e.g., the Version 0 IMS). These interfaces pose challenges in several distinct areas:

- The external systems or legacy software components submit requests which are not formatted in accordance with ECS rules. For example, TSDIS uses SFDU messages which follow CCSDS standards; the Version 0 IMS formats its requests in an Object Description Language (ODL).
- In addition, data exchanged with external systems often need to be reformatted during import into ECS, or export to the external system. For example, the Version 0 IMS expects result sets (i.e., lists of found data granules) to be formatted in ODL.

- ECS uses a distributed object infrastructure in which service requests are exchanged as distributed “object method” invocations using DOF. TSDIS and the Version 0 IMS, on the other hand, uses tcp/ip sockets to communicate with ECS.
- ECS uses DCE-based security. All requests are verified using DCE Access Control Lists (ACL) as described in the Security Section of this overview. Users of ECS need to login to their password-secured ECS account before they can obtain any services from ECS. Some Release B external systems and the Version 0 IMS do not currently use DCE authentication and security.

In Release B, ECS will be implementing an enhanced "Version 1" protocol which provides EOSDIS users with access to and extended set of information services. The ECS gateway will provide the translation of the V0 protocols for user queries and query responses to the equivalent ECS "Version 1" protocol equivalent. Please note that through the gateway, users will be limited to the suite of services provided by the Version 0 system.

As part of the gateway security effort, we will be supporting access from Kerberized clients to an ECS gateway. This will allow the ECS system to more securely authenticate the gateway users.

The ECS interfaces with external systems are called gateways. The responsibility for data access gateways belongs to the Data Management Subsystem, whereas the responsibility for data ingest gateways belongs to the Ingest Subsystem. However, all gateways pose similar design issues which can be grouped according to the above areas:

- Application level issues. The incoming request must be parsed and interpreted, data references must be translated from the vocabulary of the external system into that of the ECS data model, and the ECS objects and methods must be called which are needed to service the original request.
- Communications level issues. There needs to be support to transition from the external communications method (such as sockets) to the ECS internal communications infrastructure (DOF).
- Security level issues. An ECS login must be performed before the request can be passed on to ECS. Security policies which may be specific to each external interface determine how the gateway will verify the identity of the external users, what ECS account it will use to submit the external request, and what kind of other protection mechanisms it will employ to safeguard the external communication, as well as the ECS system.

ECS has decided to base all gateway designs on a generic blueprint called the “Gateway”. Elements of that gateway were first implemented in Ir-1, and has been refined by Release A and Release B. The generic blueprint pursues the following goals:

- Clearly separate applications, security and communications aspect of the gateway, such that they each can be re-used independently, and also to simplify implementation.
- In particular, standardize the interfaces for the security gateway, such that if the security policy for an external interface changes, the changes to the security gateway software are transparent to the communications and applications gateway components. Such changes in security policy are likely as ECS migrates through its various releases (e.g., due to a change in the security sensitivity levels assigned to ECS by NASA).

The following sections first provide an overview of the generic gateway architecture, followed by an example in which the external communication uses Kerberos.

6.3.2 Gateway Architecture

Figure 6.3-1 shows the general gateway architecture. In the figure, the gateway provides an interface between an external application (not shown) and some ECS Service.

The architecture shows the following six interfaces and components:

1. The communications gateway interfaces with the external system or software (Interface 1). From an application perspective, this interface implements a standard polling or non-polling mechanism to obtain service requests (for examples and details see the design of the Ingest subsystem), but its details are otherwise irrelevant. When a send or receive is permitted, and what is sent or received (e.g., DANs, Ingest Requests, TSDIS Status Requests, etc.), and message formats and layouts are part of the applications protocol and are of no interest to the communications gateway.

   The communications gateway is configurable (e.g., communications method, communications addresses, directory locations, mailbox addresses, etc.). The communications gateway may remove information from incoming messages which is only relevant for communications purposes.

2. The security gateway might process all incoming and outgoing messages, only incoming ones, or only the first message (via Interfaces 2 and 3). For performance reasons, the security gateway may remove itself from the event path, in which case communications gateway and applications gateway may communicate directly (via Interface 4). The security gateway may remove information from incoming messages which is only relevant for security purposes.

   The gateway will establish a DCE identity for the subsequent request and data exchange, in general after verifying the identity and authority of the external user. The specifics of this depend on the security policy implemented by the gateway. For example, the Version 0 IMS gateway will verify the user name and password, and then use it to log the external user into DCE.

3. The applications gateway decodes an incoming message, determines the nature of the request to be issued to ECS, creates the appropriate ECS client objects, formats and issues the appropriate client methods in accordance with the API of the particular ECS service which is the target of the request. The Application Gateway should call the CSS Security Services to verify that the user (now identified as an ECS account) is authorized to issue the gateway request. Interfaces 5 and 6 are normal ECS internal interfaces.

4. The applications gateway receives outgoing notices or responses from the ECS service via the corresponding client objects, reformats these in accordance with the applications protocol governing that external interface and sends them to the target recipient via the Communications Gateway (via Interface 4), or perhaps via the security gateway (via Interfaces 3 and 2) depending on security policy as controlled by the security gateway.

The presence of the security filter must be transparent to the application. That is, the interface between the communications gateway and the applications gateway (Interface 4) will be identical to that between the security gateway and the applications gateway (Interface 3).

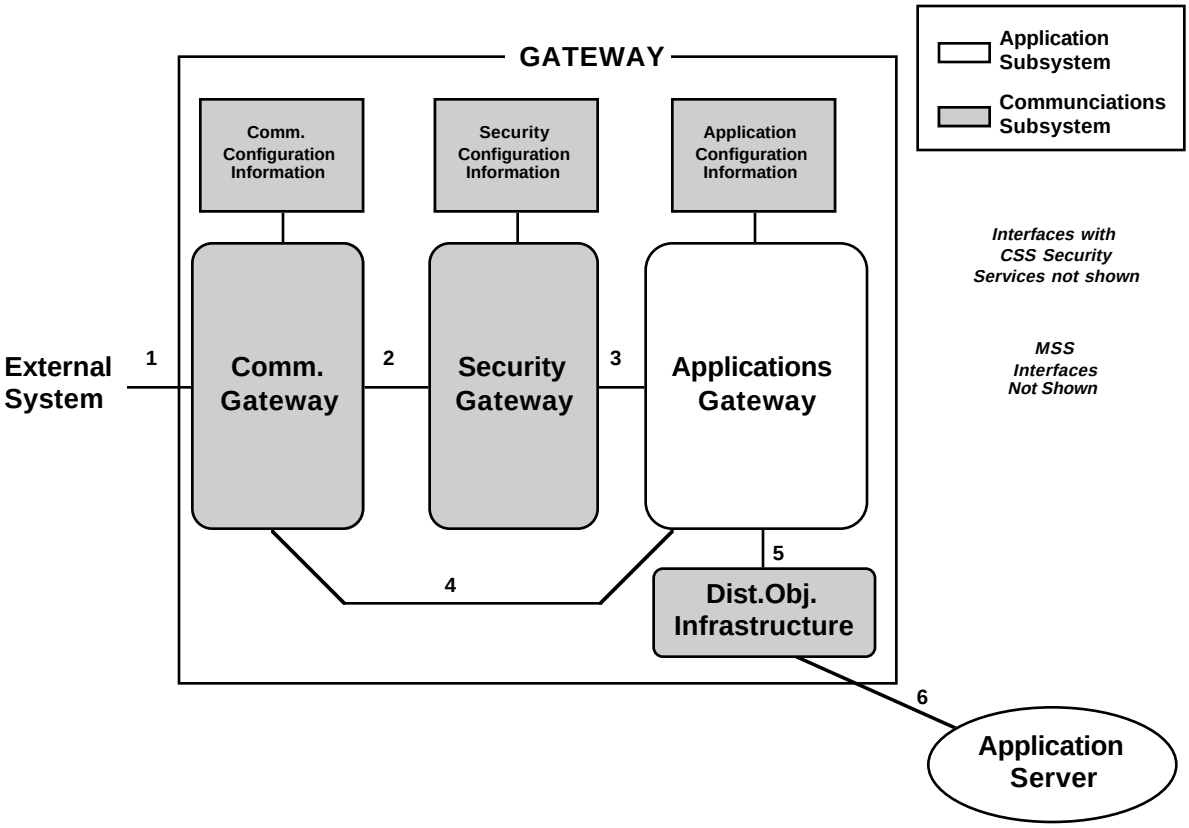

Figure 6.3-1. Gateway Architecture

Within ECS, the responsibility for the communications and security gateway is allocated to the communications services subsystem (CSS), whereas the applications gateway responsibility falls into the area of the Ingest Subsystem (INS) for ingest interfaces, and the data management subsystem (DMS) for data access interfaces.

6.3.3 Gateway Example

This example focuses on the security gateway to illustrate how different "plug-in" security gateways can implement different security policies. In the example, the external system has Kerberos installed. ECS will maintain user accounts for the users from the external system in both DCE and Kerberos Security databases. The functions of the security gateway then are to:

- authenticate the external users (validate the credentials of the principal coming into the gateway),
- authorize the external users (OPTIONAL, so only users with proper privileges can make the requests to certain ECS applications)

- provide data integrity (OPTIONAL, make sure that the data coming from the external entity is not modified in transit)
- provide data privacy (OPTIONAL, to protect the contents of the data coming from external entity from eavesdropping).

The resulting gateway implementation is shown in Figure 6.3-2. This design refines the general architecture as follows:

- The external application interfaces with the local Kerberos client, which in turn interfaces with an ECS operated Kerberos server to obtain a Ticket Granting Ticket (TGT) and subsequently, a Kerberos ticket to access the ECS gateway.
- The ticket is included in a message sent to the ECS resident gateway via a tcp/ip socket. The message is accepted by the Communications Gateway and passed on to the Security Gateway (note that the first message will be handled by the inetd, which will assign a port to this communications session).
- The Security Gateway verifies the ticket and establishes the external user's identity. It uses a mapping policy to determine the user's ECS identity - the policy might be to use the exact same identity, or there might be a substitution rule or table for all or some accounts. The security gateway then logs the account into DCE. The subsequent ECS requests which will result from this particular communications session will be submitted under this ECS identity.
- The first message exchange was dedicated to the authentication process. The application gateway is unaware that it took place. The security gateway now removes itself from the interchange. Subsequent messages will pass directly from the communications gateway to the application gateway (but only, if the initial authentication was successful).

It is easy to see how the security gateway design could be modified to require authentication for each incoming and outgoing message or to add/remove cryptographic checksums in order to ensure the integrity of the transferred information without affecting the application gateway.

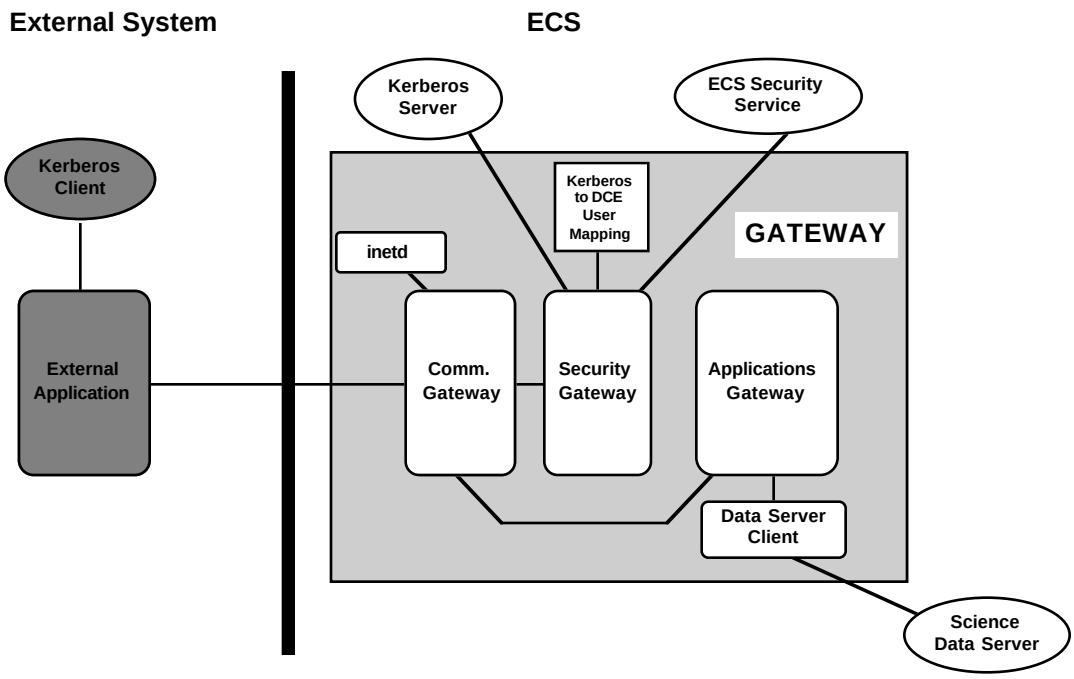

Figure 6.3-2. Kerberos Security Gateway Example

6.3.4 Interfaces to External Data Providers

ECS has a several requirements related to external data providers. These requirements mandate that ECS: be capable of supporting access to heterogeneous services; provide ECS client access to non-ECS services; permit users to invoke combinations of ECS and non-ECS services; enable ECS to operate as a federated unit within the broader GCDIS environment; enable interoperation with International data systems; and provide configuration controlled Application Program Interfaces (APIs) that enable ECS components to be reused within the GCDIS environment.

Different agencies and individuals will have differing objectives and resources that will dictate their willingness or ability to support system-wide services. The ECS architecture, therefore, must enable support for a wide range of external data providers that may exist in the GCDIS and UserDIS environments. Some of these providers will be willing and capable of supporting ECS system-wide services (e.g., distributed search) while others will not.

The ECS architectural concept is shown in Figure 6.3.4-1. It can be divided into three layers: the client layer, the service provider layer and the intersite layer. Individual sites, which may host one or more of these layers, are heterogeneous and autonomously managed. The user layer is characterized by client environments, which may be interactive (e.g., workstation graphical interface, World Wide Web interface) or process environments (e.g., analysis algorithm). The intersite layer is characterized by a set of distributed services which assess user needs against service offerings and connect the user with appropriate service providers. Finally, the service provider layer is characterized by organizations who choose to provide a set of services related to

data collections or to computer resources that they can offer. This includes the traditional data center concept and also extended data providers, whether commercial or government related (e.g. education specialists). Since the service provider layer must allow autonomous management and development, the details given here are limited to those which allow sites to interoperate. The architectural concept then, is in essence the interoperability infrastructure (the Intersite Architecture) and how the user and data provider services interface to this infrastructure.

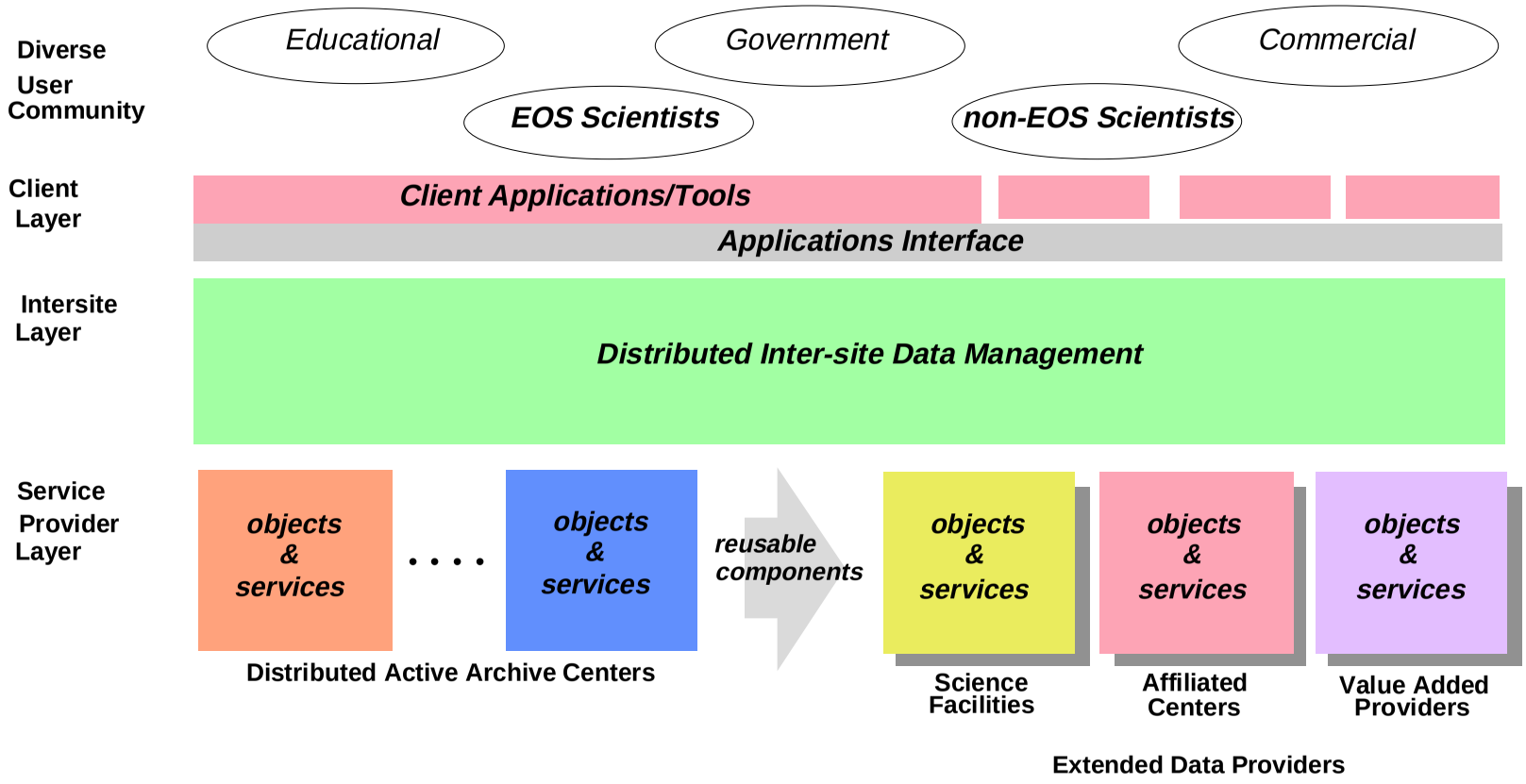

Figure 6.3.4-1 ECS Architectural Concept

6.3.4.1 Intersite Architecture

The intersite architecture is necessary to connect the services offered by providers to needs expressed by users in terms of requests to the system. Three classes of software component reside in this layer:

Advertising Service: The services offered by providers on the network need to be advertised to users. The advertising service is used by other components in the intersite architecture to perform their function.

Data Dictionary Service. This service provides access to the descriptions of data items and objects as used by one or several of the data access services advertised on the network. This service is invoked by the user or other services to correctly formulate or validate a service request.

Distributed Information Manager: Where multiple sites are needed to resolve a request then an intersite search service is required to manage the process. The service will break up the query if necessary and generate a plan containing sub-requests which will be processed at individual sites. The sub-requests will generally be characterized by queries and operations, where an operation is usually some manipulation of results to provide output in the form or context requested by the user.

6.3.4.2 Provider Interfaces to the Intersite Architecture

Each provider site chooses, or is mandated by its management, to provide certain services to the system community, or some part of that community. In many cases a service would be related to data set(s) held at the providers site, but it is not mandatory; a service potentially could access data held at another site, or even provide a service for data passed to it from another site as part of a request.

The main issues to be resolved in the interfacing of a provider's services to the entire system are briefly described in this section; they are:

- autonomous internal organization
- advertising services to users
- support for searching
- support for notifications to users when new data and/or services are available - subscription services
- support for incompatibility management
- the role of data servers.

Provider sites should be allowed to *autonomously organize* and manage their internal services and data to permit political and technical flexibility and therefore a definition of how the provider organizes their data and services is not part of the architectural concept. What matters is what the site offers for external access and how it can be accessed externally. The following discussion defines how the provider architecture is represented to the system.

Sites *advertise* their services to the system; services which are not advertised do not exist from the system perspective. The advertisements are managed by the advertising service within the interoperability infrastructure and describe what the service is and how it can be accessed. Since data is essentially equivalent to a service (a user can't access data at a provider site without some sort of service), advertisements can refer to data and services.

Sites which offer *data searching* must have an external interface for accepting searches (e.g. from the Distributed Information Manager or directly from a client), and a service for processing these requests. This is called the Local Information Manager, and is equivalent to the Distributed Information Manager, in that it resolves inexact search requests into exact queries which can be placed on individual data servers. This might involve some interaction with the user.

The nature of the architectural concept described here will result in potential *incompatibilities* between the user tools in the client layer and the services provided. Characteristics of data, tools, services, etc. which are essential for determining incompatibility are captured in an 'interoperability profile' which is stored in the advertising service. This profile is specific to the type of object under consideration (e.g. data format profile). Using these profiles it is possible to warn users of potential incompatibilities, and offer advice on how to mitigate them.

Conceptually data at a site is organized into one or more collections of related items. Each collection forms a data set which will contain both data and meta-data, the discrimination between these being provider and data set specific. To each data set one or more services are attached; the services may operate on all types of data in the data set or only one part of it, e.g., a relational database management system (DBMS) to an inventory table. Descriptions of these services and the parts of the data set they operate on are passed to the advertising service, establishing a 'data scope' against which users requests can be evaluated against. These services are called 'type' services in the concept since one type of service may be related to all data of that type (e.g., a text query service could be used for all text data within one data set and across all data sets at one site). Careful design of the type services should mean that they are adaptable by other providers for similar data. It is possible that a particular type of data would have more than one 'type' service associated with it, (e.g., two different text query services to support different access protocols) and that a type service would deal with more than just a single data type (e.g., an Object-Relational DBMS could deal with inventory data and the associated browse data).

Since extended data providers may have different data organization concepts and may employ different data management and storage technologies, the service provider layer includes a Gateway component that performs the mapping from the ECS environment to the extended data provider's environment. The Gateway component performs protocol translation, security mapping, application level protocol translation and database schema mapping.

6.3.4.3 User Interfaces to the Services

As for the provider layer, the architectural concept does not mandate how the user interfaces should work, only that they are compatible with one or more of the protocols that the interoperability infrastructure supports. The GCDIS / UserDIS concept seeks to encourage community development of components, and it is likely that the client layer is one of the areas where this will have the most impact.

Three categories of access to the interoperability layer are considered in the architectural concept:

General Access Interfaces: These interfaces are applicable to a group of similar services across all data in the network (e.g. search and order). This type of interface will be the main access to the services of EOSDIS and many of the large archives. The general interfaces can be customized in their operation by specific information from the service provider (e.g. vocabulary), etc.

Specialized Access Interfaces: The above approach would support dynamic modification of an existing interface, but would not support special interfaces for specific services, e.g. an interface which is particularly oriented towards the coincident location and analysis of sea surface temperature 'images' and sub-surface profile measurements of temperature and salinity. In this case the service provider provides a software module which provides a completely specialized interface, configured to the specific service/provider. The software module is stored in the advertising service along with the provider's service advertisement. The software module can be dynamically downloaded and configured on a client workstation, after which the user can invoke the service.

Object Access Interfaces: Finally, objects resulting from previous queries should be capable of initiating further service requests. For example a search of an image inventory might result in a results object which contains the inventory records matching the query and a reference in the object which would enable a user to automatically initiate a browse service and review the image being referenced by one of the inventory records.

6.4 System Management Architecture

ECS is a large, distributed, heterogeneous system consisting of many off the self (OTS) and developed components. The management and operation (M&O) of such a system poses numerous challenges. This section reviews the architectural concepts which are the foundation of the ECS system management support and which are applied across ECS subsystems. The section is organized as follows:

- Sections 6.4.1 and 6.4.2 review the various levels of system management and the geographic distribution of management responsibilities
- Section 6.4.3 explains how the system management responsibilities are divided up between the ECS subsystems
- The most important subsystem in this context is the Management Services Subsystem (MSS) within the CSMS segment. Section 6.4.4 provides an overview of its functions and its relationship to the application subsystems.
- Section 6.4.5 discusses an important aspect of system management and MSS, namely the handling and management of errors and faults, and the roles which the applications subsystems and MSS play in it.
- Finally, Section 6.4.6 describes how these concepts combine to support management and operations reporting.

6.4.1 System Management Levels

ECS provides system management at several levels:

- Individual Service Level - certain types of services require specialized operator knowledge. For example, a DBMS administrator is expected to know database technology in general and the specific product (Sybase) being used by the system; the manager of the science processing environment needs experience with UNIX batch production environments as well as some familiarity with various groups of science algorithms and their interdependencies

- Subsystem Level - while the smooth operation of the various services which make up a subsystem is important, it is the subsystem as a whole which provides an essential and integral set of ECS capabilities. For example, the Ingest Subsystem is needed to load and archive level 0 data. Individual component failures are of interest insofar as they impede the ability of the Ingest Subsystem to perform its function.
- Site Level - many M&O functions, on the other hand, are best executed on a site-wide basis, for example, the health of computing platforms, their operating software and applications, peripherals, and networks and networking devices. This reduces staffing requirements, and the maintenance of site-wide awareness of overall system performance and status makes for better planning and better management decisions in emergency situations.
- ECS Level - monitoring and management of the ECS mission, its budgets, and resources, on the other hand, requires an overall system view. This is also referred to as enterprise level management.

6.4.2 Distribution of Management Functions

Geographically, system management occurs at the individual DAAC, and at the SMC at GSFC, which performs Enterprise Monitoring and Coordination (EMC) functions. As a rule, service, subsystem, and site level management are performed at each DAAC; enterprise management occurs at the SMC. Table D-1 in Appendix D lists the various types of management services, and shows where they are performed or how management responsibilities are allocated between DAACs and SMC.

6.4.3 Division of Responsibilities

The ECS subsystems have distinct responsibilities for the management of various resources within ECS. In this context, the term “managed resource” refers to the following types of objects (they are also called “Managed Objects”):

- networks and network devices
- computing platforms and their peripherals and operating systems
- other special devices with their operating software (e.g., archives)
- off-the-shelf and custom developed application software services

As a general rule, each individual subsystem has the responsibility to provide system management functions at the service and subsystem level for its custom software services, any off-the-shelf software services, and any special devices employed by that subsystem. Subsystems are also responsible for supporting site and enterprise level system management through appropriate interfaces with the ECS system management infrastructure, unless such interfaces are provided by a vendor as off-the-shelf items.

Examples of service and subsystem level managed resources include databases and their database management systems, ECS application services such as data server search and access services, data distribution services, and ingest services. It is the responsibility of the respective subsystem to support the operator positions which are specifically assigned to support that subsystem, such as an Ingest or Distribution Technician or an Archive Manager. Some off-the-shelf products are used across (or by) several subsystems. Support for the corresponding operator positions is generally

provided by the OTS product vendor, however, any additional support functions which must be tailored to the specific needs of a subsystem are the responsibility of that subsystem.

MSS, on the other hand, has the responsibility for providing site and enterprise system management functions. This includes the management of networks, network devices, hardware platforms and their peripherals and operating software, and any off-the-shelf components for which the vendor provides interfaces into the ECS system management infrastructure.

Examples of system resources that MSS manages include routers, hubs, communications links (FDDI, Ethernet, and WAN links), hosts, applications, processes, operating systems, logical devices, software libraries, file systems, and peripheral devices. In managing these resources, MSS is responsible for accountability management (maintaining user profiles, maintaining an audit trail of user actions on each managed resource, and maintaining a data audit trail of actions performed on a data item); configuration management (maintaining the resource baseline, managing software changes to managed resources, and tracking change requests for managed resources); fault management (identifying, isolating, and resolving faults detected on a managed resource); performance management (monitoring both real-time and long term resource performance), and security management (protecting managed resources from security intrusions and controlling access to managed resources).

In the management of resources, the subsystems may also need to deal with security issues such as the authorization rights of a client to access a requested service or resource, the detection, reporting and possible recovery from errors (such as errors in data, or errors with media in dedicated hardware such as the archive server). Application-level performance metrics monitored may include items such as the number of data products that have been ordered, the number of data sets that have been ingested, and the number of data products that have been processed. In order to assist the subsystems in gathering this management data where necessary, MSS provides the subsystems with the capability to log the information through the use of its management agent services. This information can then be used directly by the other subsystem or imported into the management database for reporting via the report generation application. Figure 6.4-1 shows, at a conceptual level, the flow of data and commands between application subsystems and MSS.

Each of the application services will interface with MSS through an MSS provided interface. Applications provide information about themselves (e.g., current state and performance statistics), report noteworthy events (e.g., errors and faults), accept management called “instrumentation” (e.g., shut down commands), and receive notifications of events to which they need to react. Application platforms (and devices) also contain system management “agents” which are responsible for providing similar functions for the platform and its operating software (or for the device).

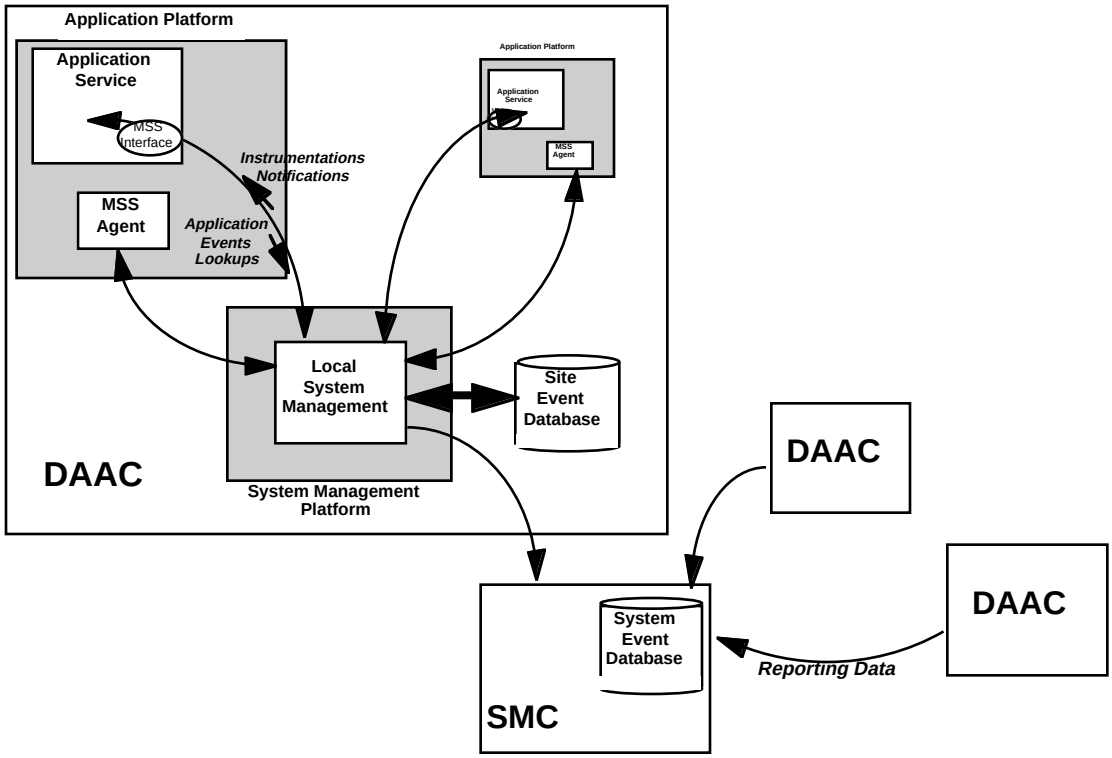

Figure 6.4-1. System Management Data and Command Flows

Events are logged at three levels. They are logged locally initially (i.e., on the application platform). These logs may be managed by the application (e.g., an ingest log), or they may be managed entirely by MSS (if the application itself has no direct interest in the event log). On a regular basis (or as needed), the event reports are consolidated into a site event history database (managed by Sybase), which can be used for ad-hoc and regular site level management reporting (see Section 6.4.5). Finally, extracts and summaries of the site event data will be forwarded to the SMC for consolidation and integration across the ECS. Logging is discussed in more detail in Section 6.4.5 on reporting, because it forms the basis for the management reporting functions.

SNMP has been chosen as the management protocol since it is the defacto and Internet standard protocol for network management in TCP/IP environments. Applications, on the other hand, interface with MSS via their MSS interfaces using OODCE.

6.4.4 MSS Overview

The most important component in the overall system management architecture is the MSS subsystem of the CSMS segment. Is has the responsibility for coordinating system management at the site and system level. The following were key drivers in its design:

- there will be no single point of failure
- DAACs will be able to manage their own resources
- it will be possible to perform system-wide monitoring and coordination

- the architecture is neutral with respect to system management policy, for example, how enterprise management functions are geographically partitioned
- within an implementation, it will be possible to distribute authority based on policy
- system management will not interfere with the accomplishment of operational DAAC functions

Avoiding any single point of failure is paramount to good system engineering practice. All system functions must be implemented in a manner that provides high availability and failure protection. The concept of a central monitoring and coordination function (versus central management and control) allows the SMC to monitor activities between all DAACs and within DAACs to identify problem areas and coordinate solutions as required. Performance analysis of the total system would become a predominant EMC function. Flows outside of ECS would be monitored at the SMC.

Local management at the DAACs of their unique resources, with visibility when required across the system is synonymous with the concept of federation in advanced enterprise management solutions. The federated approach to systems management allows the DAACs to work problems between themselves without having to have GSFC involved unless appropriate. This represents an expedient approach to problem resolution, by allowing the involved parties to directly resolve a problem.

A policy neutral architecture defers policy decisions out to the implementation (design) stage, and allows flexibility for system reconfiguration to reflect policy changes over time. Distribution of authority provides flexibility for monitoring and coordination backup, both at the DAACs and at the SMC. Finally, the concept of systems management being unobtrusive to in-line operations acknowledges that systems management is a *service* to users.

Figure 6.4-2 expands on Figure 6.4-1. It illustrates the MSS internal management data flows and the flows to ECS applications. While individual subsystems are independently responsible for process and task management-related aspects of their own subsystem, MSS is responsible for the management (monitor and control) of ECS resources, including networks, systems, and applications. Health and status of ECS resources are monitored in addition to functional areas of performance, fault, accountability, and security. MSS controls the real-time configuration of networks, systems and server applications, including system startup, shutdown, suspend, and resume operations.

The figure provides more detail on the four general forms of intercommunication between MSS and the other ECS subsystems - event logging, notification, lookups, and commands (called instrumentation). Event logging is a one way exchange from the ECS subsystem to MSS to log key subsystem level events into the MSS for monitoring purposes. Examples of application use of the MSS event logger include the start and stop of product generation, and data delivery completion. Notification is a one way exchange from MSS to ECS subsystem applications that are used to notify ECS applications of system health and status concerns that may affect the applications operation. An example of a notification is the message delivery of production string unavailability due to hardware error. Lookup is a bi-directional interface, called by the ECS subsystem applications, to obtain system information from MSS. An example of a lookup is an application request for user profile information for a shipping address to satisfy a product order.

Instrumentation is a bi-directional exchange initiated by MSS to the ECS subsystem applications to control applications processes. Generic instrumentation requests include application suspend and resume operations. Application specific instrumentation requests include requests such number of orders processed, number of browses, number of searches, or number of products delivered.

Monitoring of ECS subsystem applications is performed one of two ways. The first method is to collect information about managed objects. This is done through either UNIX commands or other utilities described later in this section. A second method of monitoring is to collect information from managed objects through instrumentation of the managed object. In this case, applications must be developed to provide information on events that generate management data and respond to MSS instrumentation requests.

MSS control of ECS subsystem applications is only through instrumentation requests. In order for instrumentation to function, the ECS subsystem applications must be instrumented to respond to MSS control requests.

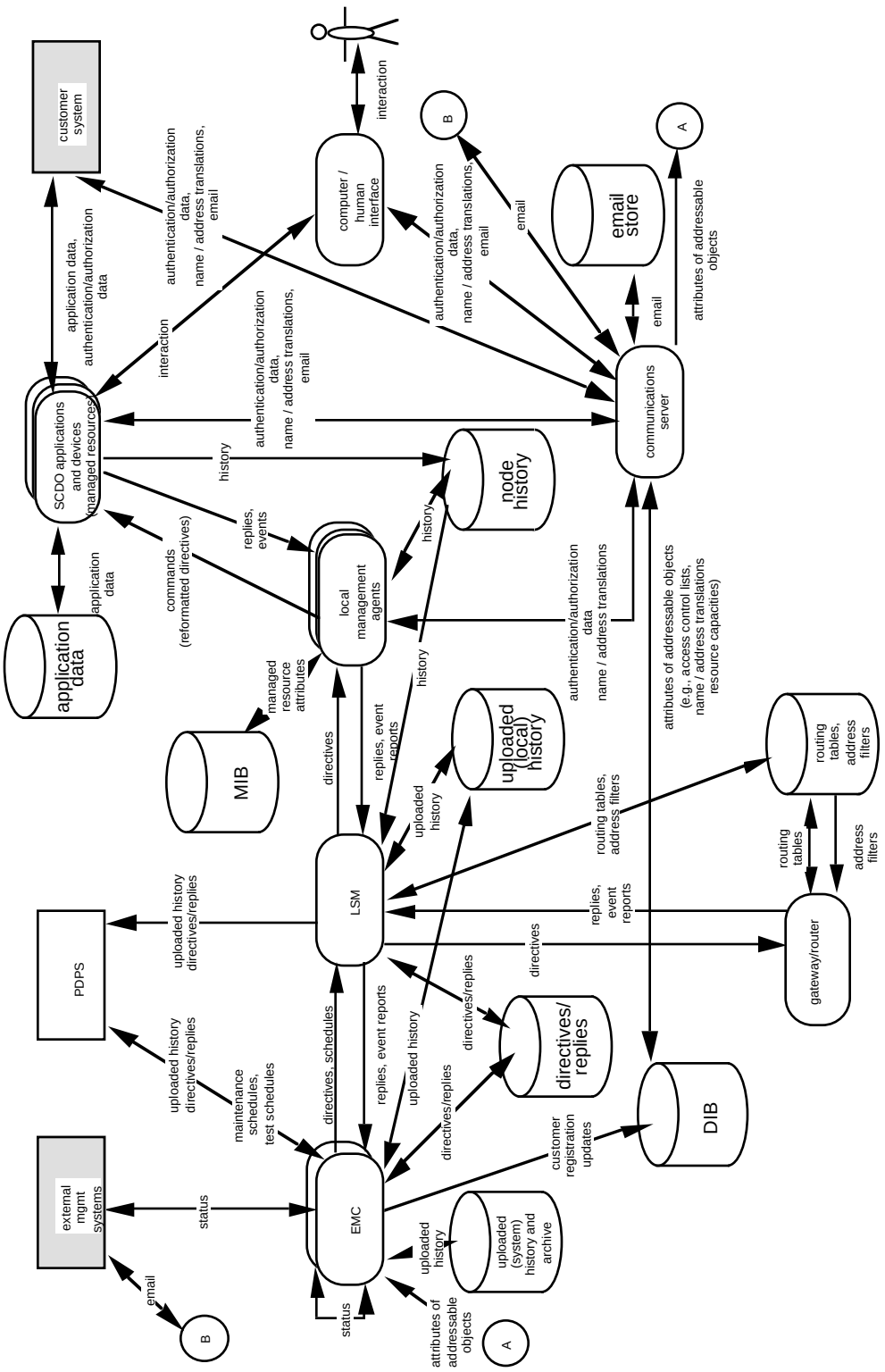

Figure 6.4-2 MSS Management Flows

The problem of the management of widely distributed resources is solved by using a manager-agent architecture, as depicted in Figure 6.4-3. The manager-agent architecture makes a formal split between two types of functions: managers and agents. Manager functions consume management information and control/initiate management actions with a managed object (resource) as the target of the operation. Agent functions, which are co-resident with the managed objects (resources) produce management data and take action on behalf of managers. That is, the manager makes management requests of the agent, and the agent emits responses to those requests. Agents act as an intermediary between the management applications and the managed objects. Further, agents are also capable of emitting event notifications to the manager. The distinction here between responses and events is that responses from agents always match up to a manager request, while events have no corresponding manager request.

The MSS Management Agents provides the following functions:

- Enable the management applications to retrieve and to set managed object values.
- Perform local polling on remote hosts to monitor the state of managed resources.
- Handle event logging and notifications.
- Provide instrumentation API to application developers to enable the manageability of ECS applications.
- Define the managed object model to represent the management characteristics of ECS applications.

Figure 6.4-3. Manager/Agent Architecture

MSS agents are based on the use of a Management Information Base (MIB). MSS requires that on each managed host, standard SNMP MIB II, Host Resource MIB, and the MIBs of network devices are supported by vendor agents. In addition, a managed object model is defined by MSS for ECS applications in SNMP MIB format. The information contained in the MIB is composed of different types of attributes: configuration, performance, fault, dynamic, static, and traps.

HP OpenView and Tivoli Management Enterprise have been chosen as the management/enterprise frameworks.

6.4.5 Event, Exception, Error and Fault Handling

One of the main areas in which applications subsystems and MSS work together is the handling of events, and in particular of errors and faults. Section 6.4.4 described how applications and MSS in principle interface, and how their responsibilities are partitioned.

This section provides a general overview of Exception, Error and Fault handling across subsystems within ECS:

- Section 6.4.5.1 presents a framework for the classification of error and fault related events.
- Section 6.4.5.2 describes rules for error and fault handling. In the ECS design, the two combine to define a set of common software objects for reuse across subsystems which provide error detection and reporting, standardize status information status information returned from object invocations, and place some application level service on top of the MSS provided infrastructure.
- Section 6.4.5.3 lists and provides brief descriptions of the various classes of faults and error conditions within ECS.

6.4.5.1 Classification Framework

The following provides the definitions for a set of terms and their related software objects in the ECS design.

6.4.5.1.1 Exceptions

An exception is an abnormal condition arising from the environment in which a program executes. An exception may be raised because of an illegal execution of a particular instruction, or it may be triggered by the application itself to report an error. For example, dividing a number by zero would raise an exception.

An exception must be handled by an exception handler. In other words, an exception will propagate upwards the calling chain until an exception handler is found which will deal with the exception. It may or may not be possible to recover from the exception within the program, i.e., the application may revert to a normal state after dealing with the exception, or the program may terminate abnormally.

6.4.5.1.2 Faults

The term “Fault” is defined in the context of the MSS concept of “Managed Objects”. A Managed Object is an system or application resource which is monitored and managed by MSS (see Section 6.4.3). A fault is an unacceptable change in the state of a managed object that leads to a change in the real time configuration of the system. A fault is typically triggered by some software

or hardware failure which is detected by or reported to the managed object (e.g., in the form of an error return status on some system call); concurrently and independently, the fault may be also be detected by an MSS agent monitoring the resource which failed. As a result of the fault, the system may stop to operate, or generate errors and then continue to run in a mode that is not considered normal (downgraded mode). There is usually a symptom before a fault occurs.

- Example 1: A parity error can kill a process (the process being killed in response to a parity error is a fault) which might have been caused by the failure of a microscopic capacitor to hold a charge, which is reflected in a bit being in the wrong state ("off" rather than "on") (symptom).
- Example 2: When one component (object) requests service from another component, service may not be provided because client has made a valid request to the server, but the server is unable to fulfill the request (due to unavailability of resource - a hardware fault)

It is possible to create fault hierarchies based on a number of categorization factors. Faults can be classified based on

- their nature (e.g. hardware fault, software fault, etc.), or
- from an operations perspective.

From a system management perspective, the latter classifications are more useful and has been adopted by ECS. For example, operational categorization will include the duration of the outage; the level of service that will be available until the fault is repaired and the system has returned to its normal operating state; the nature of the recovery actions which are needed (e.g., automatic vs. manual); and the impact on the rest of the system (e.g., localized, affecting the whole subsystem, affecting a site, or affecting ECS as a whole).

6.4.5.1.3 Errors

An error is an unexpected problem internal to the system which usually manifests itself, eventually in the system's external behavior. It is a deviation from the correct result -- an indication of the occurrence of a fault and may be reported by an exception. The error is usually returned in a status variable. An error may be detected by hardware (e.g., illegal instruction executed, arithmetic overflow), by a run-time support system (e.g., array bounds error), or by the application itself (e.g., checksum error). In other words,

- Example: When one component (object) requests service from another component, service may not be provided because the client has made a request that is outside of the server's specification. In this case, the server is unwilling to provide the service, and the client is the party in error.

Errors are associated with an error explanation (which may be reported to the operator when the error is displayed on a console), and require some recovery procedure in the software which receives the error notification. For example, if an unexpected input occurs, the common recovery procedure would be to skip the record and process the next record. Similarly, the recovery for a duplicate message is to ignore it, while the recovery for a missed message could be a request to resend.

Within ECS, errors are categorized based on the commonality of their error messages and recovery procedure. Each class of error is represented in the design as software object class which encapsulates the desired behavior and information.

6.4.5.1.4 Events

An event is an individual stimulus from one object to another or from the environment to an object. Events include error conditions as well as normal occurrences. Events are atomic and asynchronous. A function call is an event, and the return from that call is a second event. Error is also an event. Only our interpretation makes it an error. Other examples include reception of a command or a message, starting or stopping an activity, creation or deletion of an object, causing a change in the relationship, etc.

6.4.5.2 Error, Fault and Event Handling

The following describes the general framework for handling events within ECS. It includes the detection of events, their identification, reporting, and resultant actions (including entering into or recovering from a degraded mode).

6.4.5.2.1 Event Identification and Attributes

The following are general rules which define how events are identified and what attributes are associated with them, at a minimum:

- Every event will have a unique identifier for the purposes of identification and log maintenance.
- Events will identify the subsystem and subsystem component with which they are associated.
- Every event will have associated with it the identifiers and names of the calling object and the called object. This provides a mechanism for linking events as a chain for traceability.
- Every event will have information about its disposition in the form of a status (error, fault)
- Every event will record its various classification attributes (e.g., severity).
- Every event will have a holder for a message string describing the event
- Every event will have a mode attribute which will identify it to a given mode. If the event was generated by a mode independent application, then the mode attribute will be identified as "shared".
- Every event will have an identifier that will link it to it's associated parent event.

The design provides a common base class for uniform handling of events. It will contain all the attributes and operations necessary for the communication between two objects. The common base class will log certain events automatically via the MSS event reporting interface, and provide capabilities that permit the use of other logs besides the MSS log, e.g., to support debugging/ tracing of errors, analyzing performance, generating security audits trails, determining and analyzing resource utilization.

6.4.5.2.2 Exception Handling

The C++ language provides a mechanism for raising and handling exceptions that arise due to anomalies that include user, logic or system errors. If the detecting function cannot deal with the anomaly, it raises, or throws, an exception. A function that handles that kind of exception, catches it. This function is also known as an exception handler.

In C++, whenever an exception is thrown, it cannot be ignored - there must be some kind of notification or termination of the program. If no user-provided exception handler is present, the compiler provides a default mechanism to terminate the program.

The significant benefit of using exceptions is that it significantly reduces the size and complexity of program code and eliminates the need to explicitly test for specific program anomalies. Because of its many advantages, and the fact that ECS software will be developed using C++, it is project directive to use the C++ provided exception handling mechanism as one alternative for handling errors.

The following common framework will be provided during the implementation phase for uniform handling of exceptions:

- Naming convention. All exceptions will be named according to a common convention that is tied to the underlying error being reported through the exception.
- Event / Message Catalog. All events will be collected in an ECS wide event catalog, receive a unique identification, and will have explanatory text (suitable for display to operations personnel) associated with them.
- Linking events. Links must be provided to trace the exception to the underlying error that raised the exception and the corresponding event that produced the error.
- Event hierarchy. Because exceptions are a mechanism for reporting errors, the exception hierarchy will be identical to the error class hierarchy.

6.4.5.2.2 Catch-all Exception Handler

Each application service must have a catch-all exception handler which reports other-wise uncaught exceptions. A catch-all exception handler is a default exception handler that terminates any exception which is propagated up to the top of the calling chain.

6.4.5.2.3 Fault Handling

The ECS Fault handling is performed by MSS which provides the following services:

- MSS Objects for notification, lookup, event logging and instrumentation (a common interface for handling faults through change of state of managed objects and MSS Event log)
- Fault Management including notification to dependents that a fault has occurred and launch of a recovery procedure to recover from the fault

All faults are reported, and the reported information includes a fault classification attribute. The following classifications are used by ECS:

- Minor degradation. The component (object) can continue to provide all the services that it supports, although there may be reduced performance. For example, a storage device may

fail, but the service can continue using the remaining storage devices. Likewise if a couple of processors fail in a Symmetric Multiprocessor in the Processing Subsystem, other processors can be used to make up the loss.

- Major degradation. The component (object) is unable to provide some services but can still support other services. For example an instance of a Client is unable to send or receive messages due to a network problem, but can continue to provide local services.
- Loss of services. The component (object) is unable to provide any useful services. For example, if a critical COTS component fails, a loss of service may result.

A loss of service may need a common recovery procedure like restart, or remove failed component and integrate new component irrespective of whether the failed component is a disk, processor or network. Also, such a classification scheme could introduce a common display procedure that colors an icon in the M&O GUI a very dark shade of red for loss of service and a lighter shade of red for minor degradation. In summary, faults will be classified based on operations concepts and M&O perspective.

6.4.5.2.4 Fault Tolerance and Recovery

Fault Tolerance is the ability of the system to continue to operate in the presence of faults. For example, if an archive drive fails, but the archive has two or more drives, the archive will continue to be operable, albeit at a reduced rate of throughput. When the system suffers a fault which reduces some operational system capability, the system is said to operate in “degraded mode”. When the original failure has been repaired and the system is returns to its normal operating capability, the system is said to have “recovered.

There are three main types of fault response:

- Continue to process the event despite the failure. The system may be changed to a downgraded mode until the cause of the fault is removed and the system can return to normal operation
- Abandon the event, perform any cleanup and regain control for handling new events
- Cause failure (Fail-stop - the application will either operate according to its specification or it will stop). No more events are processed until a recovery procedure (automatic or manual) is completed

ECS subsystems will be designed to continue to operate in degraded mode (rather than terminating their operation completely), whenever possible. It is also an ECS design goal to have subsystems recover automatically to their normal operational capability when the original fault is repaired. Where the application cannot be able to recover on its own, the MSS Fault Manager will launch the designated recovery procedure.

6.4.5.2.5 Error Handling

All errors will be reported to MSS via the event reporting interfaces. The reported information will include the error identification, an error message, and any addition attributes which are needed to describe the detailed nature of the error (e.g., for a file open error, the type and name of the file which could not be opened.

6.4.5.2.6 Error Detection

An important aspect of errors is the method of their detection. Within ECS, methods of detection are categorized into Detector Classes. Detector classes are derived by examining each input to the application and enumerating the types of error detection which are possible for those inputs (some error detectors may not be dependent on any specific input). The following is a basic list of detector classes:

- Response time exceeded - many device access services monitor the health of devices by using timers. To avoid redundant error detection code, individual service implementations should be examined to see if this check is done below the application level.
- Capacity exceeded - applies to all storage including datasets, queues, heaps, etc.
- Missed message - for point-to-point messages, the lower level communication protocols will check sequence numbers, guaranteeing that the messages are delivered in order to the application. This is not the case for broadcast messages.
- Faulty/Unexpected input - the input can be checked for syntax and semantic errors without reference to state data
- Duplicate message - this happens when the original component sent a message to another component and failed after sending the message, the backup component has resent the same message. The receiving component will give this error.
- Input/state inconsistency - the input, when checked against state data is found to be semantically incorrect
- Faulty state or logic - for example: directory corruption; or previously committed state data now looks bad, often due to inconsistencies among data sets, an unplanned execution flow.
- Negative response - a peer or lower level service returns an "unwilling" or "unable" response
- Heartbeat - A solicited or unsolicited periodic message was not received. Generally this occurs with external interfaces and attachments
- Stop (relayed from part of the environment) - PGE failure

  A stop can mean one of two things: a) the address space stops, or b) the processing of the particular event stops. When this behavior is desired, the failure of a particular event should not be allowed to affect other events (for e.g., partial database updates should be rolled back, or the particular database should be marked as unusable)
- Unknown ("unknown" should never be explicitly reported; it exists to account for bugs in error handling logic that leave this field unspecified)

6.4.6 Management and Operations Reporting

6.4.6.1 Overview

Operations personnel and management need to be able obtain reports about various categories of system performance. This is a very important aspect of overall system management. Many of these reports are generated in an ad-hoc fashion, usually in response to some problem or concern. For example, operations may have noticed a recent slip in the average time needed to fill orders for a

certain product collection. To investigate the cause, operations may run various statistics to find patterns which may indicate the source of the problem. The ability to produce such reports is, therefore, an important aspect of the system management architecture.

ECS provides three levels of reporting and two methods of reporting. The three reporting levels are:

- reports which are specific to the operation and management of a subsystem and are generated by that specific subsystem - for example, the physical allocation of database resources are reported directly by the DBMS for use by a database administrator;
- site level reports which integrate information across several subsystems or provide mechanisms for obtaining long terms statistics about the ECS - for example, regular statistics about the various categories of data production and distribution; and
- system level reports which integrate information across ECS sites, allow comparison of ECS operation at different sites, or support end-to-end tracking and reporting of inter-site activities.

The two reporting methods are:

- ad-hoc reports - ECS will provide tools to allow operations personnel to create reports in an ad-hoc fashion, direct them to a printing device or a file, and save such reports for repeated execution.
- canned reports - operations will have a number of predefined reports which will be executed regularly, and through their common formats allow comparison and long term trend analysis.

The three levels of reporting correspond to the three levels of logging which are performed within ECS and which were described in Section 6.3.4. Subsystems perform their own logging and performance tracking to support the personnel directly involved in operating that subsystem. For example, data processing events are logged within the Planning and Processing database, and that database can also provide information about the current processing status for each group of data products. Concurrently, processing events are reported via an MSS interface to the management subsystem. The logs generated at each host are consolidated on a regular basis with a configurable time constant (which typically would be in the order of 15-30 minutes) into a database of site-related system events. Also on a regular, configurable basis, excerpts and aggregates of this information are sent to the SMC where they are stored in a database for reporting and tracking of activities and requests which span sites.

The mechanisms offered for reporting at the subsystem level depend on the specific configuration of each subsystem. Release B subsystems provide the following off-the-shelf capabilities for this purpose. Most of these components support regular reporting through saved reports, in addition to ad-hoc reporting:

- Sybase database administration and reporting writing tools
- Autosys and Autoexpert reporting capabilities
- ClearCase and other configuration management tools
- HP OpenView reports
- Fault and performance management applications

Site and system level information is collected in a Sybase DBMS, and reporting is provided via the Sybase report writing tools and Sybase queries.

6.4.6.2 Report Data

There is a general requirement to be able to reconstruct the performance of the ECS, its operators, and its users as part of performance analysis, error debugging, and testing.

All logged items will be identified by date and time so that the sequence of events can be reconstructed. The history log will be organized such that the information can be selectively retrieved based on logical combinations of criteria such as: user, order, platform, instrument, data product, data product type (e.g., standard, interim), production mode (i.e., routine, on-request), operating mode (i.e., live vs. reprocessing vs. test), production string, Subsystem/CI/CSC/ subroutine/object, interface source and destination, and specific logging IDs.

Typical information logged in a history log database includes:

- operations data,
- resource data, and
- user statistics.

They are briefly discussed in the following paragraphs. Tables D-9 through D-13 in Appendix D present a summary of the different types of reports and their envisioned contents.

6.4.6.2.1 Operations Data

Logging of operations data is intended to establish an electronic record of key data, inputs, outputs, and faults. Suggested classes of logging are:

Initialization Data.	Each ECS service will log all initialization/control files, values, and resource status each time it initializes.
Data Transfers.	Source/destination, data identification/type, media type, and size of all data transfers to/from each application subsystem, including external and internal ECS transfers. Logging will provide enough information to associate each event with the original stimulus (i.e., a particular user's request). Logging will unambiguously identify the ECS source of the data within the subsystem, and the data destination (e.g., another ECS service, a user, or an external system). Network addresses will be included where appropriate.
Human Actions	Source (i.e., operator, analyst, user services, etc., and user) and values of human operator inputs which request changes to the state or configuration of a service.
Fault Messages	All faults will be logged. Contents, and destination of all operator and user alarms, alerts and notifications will be logged.

Intermediate Results	ECS application services will be able to log selected intermediate results of processing or activities that provide insight into the operation of the CI. This type of logging will be configurable and can be turned on or off by operations.
Product Generation	Processing history and statistics will be kept and will include information on PGE execution times, comparison to plan, the number and timing of replans, information on the data products created, PGE resource utilization, and QA attributes of the produced data.

6.4.6.2.2 Resource Utilization Data

ECS will monitor all computational, storage, archival, ingest, network and distribution resources and be able to construct short term resource utilization reports (e.g., reflecting current throughput and resource utilization rates) and long term trending (i.e., resource utilization over longer reporting periods).

CPU	CPU utilization
Computer memory	Memory utilization
Local storage	Local disk/mass storage utilization
Archives	Archive utilization (by file and science product, number of bytes in use, number of bytes in use, I/O, number and size of retrievals, number and size of storage actions
Hard media	Hard media distribution utilization (i.e., number of media by type produced, number of bytes produced).
LANs	LAN (including gateways, routers and bridges) loading and performance.
WANs	WAN traffic in and out of ECS.

6.4.6.2.3 User Data

User satisfaction will be a key metric for ECS. ECS will be able to track an order from the time it is placed until the order is fulfilled, either electronically or via hard media. ECS will collect statistics on volume by distribution type, timeliness (elapsed time from order receipt to data ready to data delivered), quality (did the user receive what was ordered), and resources utilized in servicing users. Data products will be logged to at least the granule level. ECS will be able to provide information about:

Order development	Number and nature of data product requests (e.g., if product A is ordered, product B is always ordered too); mechanism used for placing an order (electronic or through User Services); amount of data requested.
Order processing	Size and development of the order queue; elapsed time needed to respond to the various types of orders.

Order delivery	Size of the distribution queue; data products and files included in the order; size of order; distribution mechanism (i.e., electronic, numbers and types of hard media); time and resource utilization for delivery (i.e., awaiting pickup or media creation); elapsed time from placement of order; elapsed time to deliver the order; order fulfillment success rate.
Order satisfaction	Incomplete orders; rejected orders; orders never picked up; orders never shipped.
Order resources	System resources utilized by ordering, and distribution over orders.

6.4.6.2.4 Security Data

Security relevant events and statistics which are of interest to security analyses will also be collected, including:

Security Violations	Attempted connections, requests for services which failed due to a lack of authorization, and other types of attempted security breaches will be reported.
Unusual Patterns	The information collected for system monitoring purposes can also be used to examine logged events for patterns of activities which might indicate attempts to compromise ECS security.

6.4.6.3 Reporting Capabilities

Although ECS will include a set of canned reports, it was decided not to make these reports part of the formal system design. The decisions is based on the following considerations:

- Canned reports generally include less detail and more aggregates and statistics than ad-hoc reports. Other than that, there is little difference between ad-hoc and canned reports. In general, a canned report is a version of an ad-hoc report, saved (and perhaps parameterized) for future execution. ECS operations personnel will be fully qualified to create ad-hoc reports, and tailor any canned reports which ECS initially provides over time to their needs.
- Making the canned reports part of the formal system design would place them under formal configuration control. This would make it much more difficult for system operations to change the regular reporting to suite their needs based on their experience with the operation of the ECS.

The reporting capabilities provided by the individual ECS subsystems are discussed in the individual subsystem design documents. However, it was felt that from an M&O perspective, the topic truly crosses subsystem boundaries, and that an integrated presentation is needed. Therefore, the remainder of this section provides an overview of the reporting capabilities which ECS will provide. The capabilities are organized into five functional areas:

- Fault Management
- Performance Management
- Accountability Management
- Security Management, and
- Configuration Management

For each of these area, a table is provided in Appendix D listing various M&O topics and the corresponding reporting capabilities. As indicated above, these reporting capabilities will be developed in an incremental fashion involving the ECS M&O organization as well as the DAACs, but ECS will develop models for these reports to facilitate their further development by M&O.

6.4.6.3.1 Fault Management Reporting

HP OpenView is the ECS enterprise management framework. Management data for all devices that are monitored via SNMP agents is collected and can be displayed by HP OpenView. The operator can select one or more icons representing the managed object(s) for which reports are to be generated. HP OpenView provides both standard reports for real-time monitoring of network devices and ad hoc reports for real time and near-real time monitoring of any management information defined in the loaded MIBs.

Fault Management reports will include summary and detailed reports on managed objects' faults, i.e., hardware, software, and network faults occurring at each site. Reports may be generated to cover all or portions of the data captured in the database for variable amounts of time. For example, the ground resource fault and maintenance reports generated by LSM at each site may include fault type and description, time of occurrence of fault, effect on system, fault resolution, fault statistics, etc., or any other parameters required.

HP OpenView provides four predefined reports for real-time fault management reporting. The Ethernet Errors and SNMP Errors reports provide statistical information about errors that occur from the time that the operator selects the report for operator-specified managed object(s). The SNMP Authentication Failures report and the SNMP Events Log provide reports of past events that have been logged by HP OpenView, again for the operator-specified nodes (although the SNMP Events Log can also be reported for the entire system).

The majority of fault reports will be generated on an ad hoc basis. The ad-hoc reports can be generated on as needed basis by HP OpenView, the fault/performance management application, the report generation application, and the trouble ticketing application to include detailed and summary information on the fault management of ground resources as required by the operator.

6.4.6.3.2 Performance Management Reporting

Performance Management has a number of different aspects. It includes the performance aspects of the various system resources, as well as the performance of ECS services with respect to their service requests. The reporting capabilities are provided by ECS performance management tools (which are still in procurement), ECS COTS tools used by the various services, and Sybase report writing tools (to extract performance data from the information in the site and SMC history log databases).

6.4.6.3.3 User Services and Accountability Reporting

These reports will be based on accountability data stored in the Sybase management database and will be generated using the Sybase report writing tool. The purpose of these reports is to track and profile ECS usage by external (e.g., science) users. The purpose of these reports is to identify usage trends to identify service needs/shortfalls and assist in the planning of future capacity needs. For Release B, this reporting area will include accounting reports.

6.4.6.3.4 Security Management Reports

Security management application report on security events for its management domain, i.e., a given site (for LSM), or the entire system (for EMC). The security management application is currently in the procurement phase (no products have been selected yet). At a minimum, the products that will be selected will provide the capability to store security management data in the management database, from which the Sybase report generator will be capable of creating standard and ad hoc security management reports. The security management application may or may not provide separate reporting capabilities. In addition, virus detection and other security reports will be provided via an ad hoc reporting capability and generated on an as-needed basis

6.4.6.3.5 Configuration Management Reporting

The Configuration Management applications (Baseline Manager, Software Change Manager, and Change Request Manager) will be COTS products and will provide functionality which will support the management of ECS resources. These products will track what constitutes ECS baselines; make available functional and physical characteristics data needed to operate and maintain the system; aid in managing system requirements and changes; and provide report generation capabilities.

Custom reports generation will be supported by the built-in capabilities of the ClearCase and other CM COTS.

6.5 User Interface Architecture

The user interface architecture for Release B ECS has three major components, science user interface, operator interface, and the common desktop environment (See Figure 6.5-1). The science user interface is composed of custom-developed X-Windows and HTML applications which support science activities such as product search and retrieval. The operator interface, which is composed of both custom development and COTS, supports operations activities such as data ingest, product processing, user support, and software configuration management. The common desktop environment (here referred to as a generic concept, not the product, CDE) is a container for both the science and operator workbenches and provides a mechanism through which all ECS application windows may be managed and accessed.

Table 6.5-1 shows the listing of ECS tools which have science user and/or operator interfaces. Each of these components are described in the following sections. The first two sections describe the functional capabilities and tools of the science user and operator interfaces, respectively. The third section discusses the capabilities of the common desktop. In addition these sections discuss examples of current design philosophy.

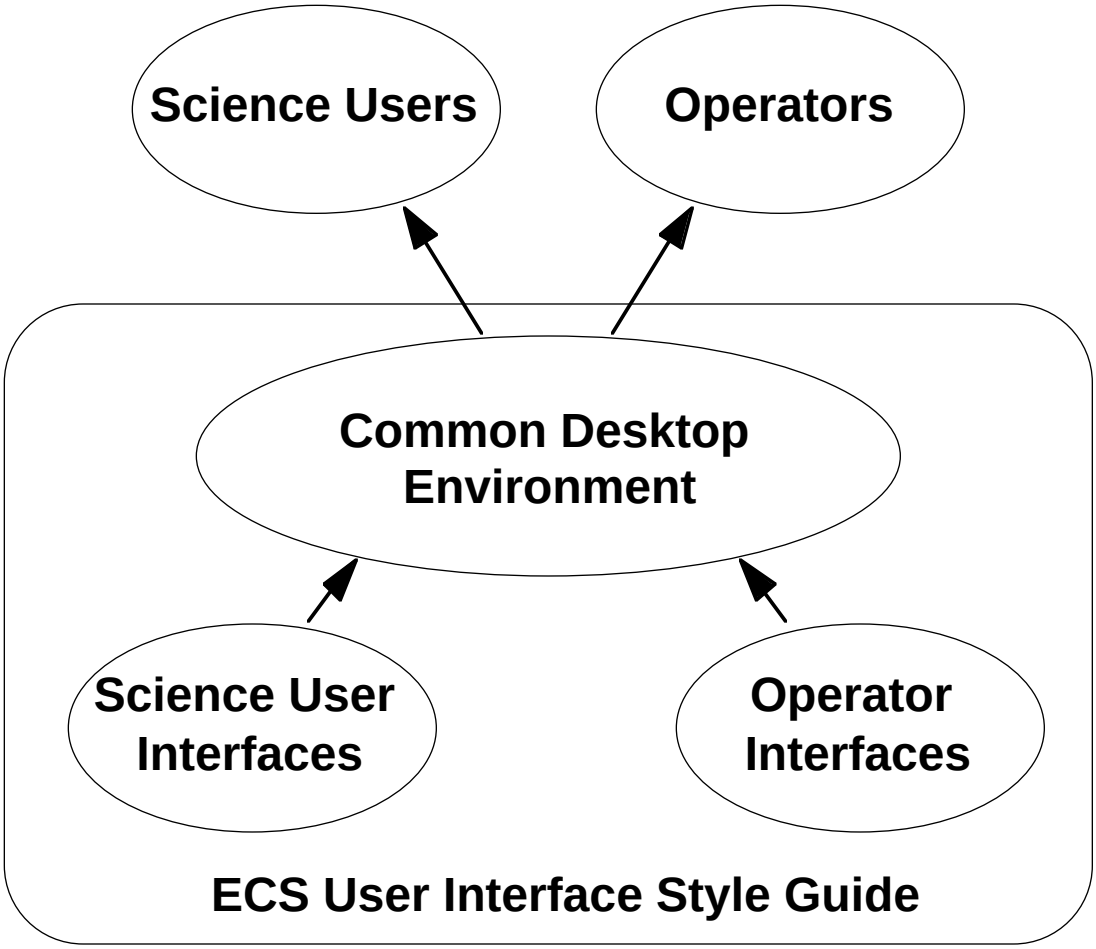

Figure 6.5-1. User Interface Architecture Overview

Table 6.5-1 ECS Tools with User/Operator Interfaces (1 of 3)

S/S	Capability	Tool	Type	User
CLS	Directory, Guide, Inventory search	Earth Science Search Tool	Custom, Motif	Science User
CLS	Product request	Product Request Tool	Custom, Motif	Science User
CLS	Browse Information	EOS View	Custom, Motif	Science User
CLS	Data Dictionary access	Data Dictionary Tool	Custom, HTML	Science User
CLS	User Registration	User Registration Tool	Custom, HTML	Science User
CLS	User Profile	User Profile Tool	Custom, HTML	Science User
CLS	User comment collection	Comment/Survey Tool	Custom, HTML	Science User
CLS	Data acquisition (Aster)	Data Acquisition Request Tool	Custom	Science User

Table 6.5-1 ECS Tools with User/Operator Interfaces (2 of 3)

S/S	Capability	Tool	Type	User
-	Science software development	SDPS Toolkit	Custom	Science User**
-	Communications and system management	CSMS Toolkit	Mixed	Science User**
IOS	Advertising of providers, services and products	Advertising Tool	Custom, HTML	Science User and Operator
DMS	Data Dictionary Maintenance Tool	DDMT	Custom, Motif	Operator
INS	Ingest monitor and control	Ingest Tool	Custom, Motif	Operator
PLS	Production plan activation and creation	Production Planning Workbench	Custom, Motif	Operator
PLS	Data processing requests	Production Request Editor	Custom, Motif	Operator
PLS	Subscriptions for data	Planning Subscription Editor	Custom, Motif	Operator
PLS	Resource Planning	Resource Planning Workbench	Custom	Operator
DPS	Execution of PGEs	AutoSys	COTS, Motif	Operator
DPS	AI&T package definition	AI&T SSAP Tool	Custom, Motif	Operator
DPS	AI&T PGE definition	AI&T PGE Tool	Custom, Motif	Operator
MSS	Baseline Management	XRP II	COTS, Custom	Operator
MSS	Software Change Management	ClearCase	COTS, Custom	Operator
MSS	Change Request Management	Distributed Defect Tracking System (DDTS)	COTS, Custom	Operator
MSS	Software Distribution Management	Tivoli-Courier	COTS, Custom	Operator
MSS	Software License Management	FLEXlm, iFOR/LS	COTS, Custom	Operator
MSS	Inventory, Logistics, Maintenance (ILM)	Note 1	COTS, Custom	Operator
MSS	Training management	Training Manager	Custom	Operator
MSS	Policies & Procedures management	DDSRV, INGST, Client, BulletinBoard	Reuse of Custom	Operator
MSS	Fault management	HP OpenView Network Node Manager (NNM)	COTS, Motif	Operator
MSS	Performance management	Tivoli and HPOV NNM	COTS, Custom	Operator
MSS	Security management (network)	Router tables	COTS	Operator

Table 6.5-1 ECS Tools with User/Operator Interfaces (3 of 3)

S/S	Capability	Tool	Type	User
MSS	Security management (distributed communications)	HP DCE Cell Administration	COTS	Operator
MSS	Security management (host)	TCP wrapper, npasswd, crack, SATAN, and Tripwire	COTS and public domain	Operator
MSS	Accountability management	Sybase	COTS, Custom	Operator
MSS	Physical configuration management	Mountain View	COTS, Motif	Operator
MSS	Trouble ticketing	Remedy	COTS, Motif	Operator
MSS	Management data access	MDA	Custom	Operator
MSS	Billing and accounting	Note 1	COTS, Custom	Operator
MSS	Report generation	Note 1	COTS, Custom	Operator
CSS/ MSS*	E-mail	OTS (SMTP)	OTS	Operator/User**
CSS/ MSS*	File Access	FTP, KFTP	OTS	Operator/User**
CSS/ MSS*	Bulletin Board	NNTP_Capable	OTS	Operator/User**
CSS/ MSS*	Virtual Terminal	Telnet	OTS	Operator/User**

* CSS provides the service, MSS provides the operator interface as part of the Software Distribution Manager.

** Distributed to operators and science users on algorithm development teams.

Note 1: These items are in final negotiation stages. This information will be finalized at CDR.

For a more detailed description of any of the user or operator interface capabilities discussed in Section 6.5, please refer to the appropriate subsystem design document (305-CD). Science user interfaces are contained within the CLS and IOS subsystem design specification; operator interfaces are contained within the DMS, DSS, INS, DPS, PLS, and CSMS design specification documents.

6.6 Earth Science Query Language and Protocols

A key ECS design driver derived from interactions with the Earth Science community and NASA is that the ECS provide a simple and easy to use, yet semantically rich access mechanism for users and their applications to access ECS supported Earth Science archives and services. The information search and retrieval approach has evolved from the search and order model used in the V0 IMS's Graphical User Interface, ODL and sockets design to the Release B V1 design based on a publish and subscribe model built on access to a Distributed Object Framework, Earth Science Data Type collections and a Query object specification.

The V1 model for search and access leads to several design approaches that are influenced by

- the need to provide flexible efficient access internal to ECS,
- the need to build on COTS software to reduce overall system lifecycle costs,
- the requirement to interoperate with external U.S. and international data centers,
- the use of relevant query and data access standards (including defacto standards such as HTTP (hypertext transfer protocol) on the WWW (world wide web).

The query language standards efforts for data access services both in the extended relational and object-oriented database management communities have not converged on a standard and are not expected to until the late 1997 or early 1998 time frame. The ECS Release B V1 design must therefore design a system based on proven technologies with a means of evolving to the standards as they converge over the next three to five years. The approach chosen by the ECS design team is to encapsulate vendor specific data access implementations within the Distributed Object Framework and capture query specifications in a Query object. The Release B strategy is to work within the ECS database management system vendor baseline and track the query specification standards through subsequent releases. The ECS team will work closely with the baseline DBMS vendors to ensure compatibility with the ECS standards requirements both in the data management (SQL3) and the distributed object (OMG/ODMG) domains. It is fundamentally important to the international community as a whole and the Earth science community in particular that a consensus be reached among the SQL3 and OMG/ODMG standards efforts.

The catalog interoperability community via the Committee on Earth Observation Satellites, the relational database vendors, and the distributed object system community led by the Object Management Group currently have inconsistent models for query languages, data models, and access protocols. The catalog interoperability community is investigating a Catalog Interoperability Protocol (CIP) that is a derivative of the Z39.50 standard. Relational database vendors are for the most part participating in the ANSI SQL3 standardization effort and include vendors such as Oracle, Sybase, Illustra, and IBM (DB2). The Object Management Group and the Object Data Management Group are sponsored primarily by object-oriented database and tool vendors and are attempting to standardize on the ODL/OQL specification.

Working groups have been formed in each of the standards communities including the ANSI X3H2 committee on Database Languages and the ODMG (Object Data Management Group) consortium leading the Object Management Groups efforts in standardizing query and persistent object services. These working groups are attempting to resolve any inconsistencies among the proposed standards at least to the level of defining consistent subsets of the standards to support open system interoperability including languages, protocols and data models.

This overview section will explain the standards influencing the design activities and the current Release B V1 design alternatives for search and access to ECS data collections and services. A set of representative queries will be described in English text and structured query language to motivate the design options and the baseline design presented in this design document.

6.6.1 Introduction and Context

Search and access to Earth Science data has evolved from simple attribute value matching as in the V0 system prototype, through table based SQL-92 based client server interaction as in local DAAC

implementations, through library catalog based search such as with Z39.50 with gateways into local databases, to ongoing attempts (by standards committees and database vendors) to integrate a full query language access to an object oriented system repository. The SQL3 and ODMG OQL standards efforts are attempting to define a query language that will provide access to complex data types and associated attributes and operations. Subgroups are investigating the standards for spatial access, temporal access, and multi-media access.

The basic ECS requirement for user access to the system is that it provide ease of use and data location transparency. The first requirement implies that the interface language (either graphical or textual) be in a form understandable to Earth Scientists and other users. Use of complex computer science terms would be unacceptable to the scientists. The use of Earth Science terms for data and algorithms that manipulate the data is therefore more desirable. Data location transparency implies that users must not have to know where data is located in order to search it. The query language must therefore be able to specify data collections independent of their location and, by implication, independent of their local schema and data format.

The ECS requirements address access to data based on geophysical parameters (which are in some cases attributes and other cases full data types), spatial constraints and temporal constraints. The ECS design for data collections is based on Earth Science Data Types (ESDT) and Computer Science Data Types (CSDT). In formulating requests on the repository the assumption in this paper is that the user or application program would draw on a set of ESDT's (defined in the Data Dictionary service) with associated attributes, methods, and operations to formulate an Earth Science Query. The query could be formed in its simplest form via a graphical or forms based interface (such as the Earth science search tool in the ECS workbench) within the Client Subsystem's desktop. More complex queries (not representable via forms or graphically) could be entered in text by users or application programs. Simple syntax sensitive editors can be used to check the validity of the constraints and the syntax of the query.

A simple example of the use of a query language supporting ECS data may clarify the design approach alternatives. The example considers a scientist searching for documents, and in particular research papers, published over a specific period of time, and addressing specific topics. The assumption is that a document, defined by a Document ESDT, has as attributes a DocumentType, Title, author, publication date, country, language, organization, an abstract, a body (structured by paragraph, section, or chapter), a set of topics (calculated by searching the document via index inversion), and a set of references. The methods performed on individual documents include categories of creation, deletion, versioning, splitting, hyperlinking, searching, and updating. Operations performed on multiple documents include merging, comparing, and cross-hyperlinking. The English language query and the structured query based on the English description appear as follows:

More complex examples will be developed to illustrate three categories of complexity in queries as developed in early 1994 during several design working groups involving the ECS team, NASA, and the Earth Science community. These more complex queries cover the full spectrum of ECS ESDTs including directories and inventories (as tabular views into the data product collections), guides, research papers, browse products, production histories, and data products. In the Release B V1 design, two key abstractions are used to communicate with the ECS, the ESDTReferenceCollector interface and the Query interface. The protocol used to access ECS is coupled with the interface to the ESDTReferenceCollector.

There are three factors related to information intensive systems interoperability with each other:

1) The language used and the associated vocabulary (syntax)
2) The data or Knowledge model definition (semantics)
3) The dynamic communication mechanism (protocols)

For example in the WWW the language used is the HTML (Hypertext markup language), the protocol is HTTP (Hyper text transfer protocol), and the data model is a simple hyper text document model. In relational databases the language is SQL92, the protocol is RDA or ODBC, and the data model is the relational model. The NASA V0 system uses a parameter value language (ODL), sockets as the underlying communication protocol, and a simple Directory / Inventory data model.

The ECS V1 system uses an object oriented OSF DCE infrastructure as its internal communication foundation. A distributed object framework defines the external interfaces to be used when accessing system services. Relating the ECS V1 design to the three factors described above, the language used is an attribute-value-service triplet encapsulated in a query object, the data model is based on the CORBA/OODCE interface model defined by the Interface Definition Language (IDL) and represented via the ESDT semantics defined in the Data Dictionary service, and the protocol is defined by the sequence of services provided on each publicly available distributed object running on top of the OSF DCE RPC (remote procedure call) protocols --- built in turn on top of TCP/IP or UDP.

The detailed design of the interface and the options chosen will be documented at the Release B CDR in early 1996. The design will evolve from that presented at the Release A CDR covering the V0 gateway interface to the ECS Distributed Object Framework and the Science Data Server and Document Data Server Designs. The ECS Release B components addressing the query services are the Client Workbench, the Advertising Service, the Data Dictionary Service, the distributed data management services (DIM/LIM), and the science and document data servers.

6.6.2 ESQL Scenarios

The scenarios relevant to the Earth science query language (ESQL) discussion are a subset of those presented in the Release B design specifications. In particular the ESQL is an attempt to structure the kinds of service requests, and in particular the search requests, made by science users. The ESQL provides formal access to the ECS data collections and services and may be issued directly by users and their applications or indirectly via ECS provided toolkits. Three examples of the usage of the query language are presented here. More complete examples are provided in the Release a design specification sections.

6.6.2.1 Simple Attribute Query (Document, Directory, Inventory)

English Text: Retrieve as hyperlinked HTML 3.0 documents the abstracts, author's name, and references of all English language research papers publish from 1990 to present that cover the topics of the correlation between sea surface temperature and the ozone hole.

Explanation: The structured query language accesses the Reference Paper ESDT and assigns a reference X to each instance. The select clause constrains those attributes of the document that will be returned to the client as a result of matching the search constraints in the Where clause. The Select clause further specifies that each resulting document will be formatted as a hyperlinked set

in HTML 3.0 format. The Where clause provides the specification of the search criteria to the document search engine.

Structured Query Language Text:

```
SELECT  (X.author, X.abstract, X.references).hyperlink("HTML 3.0")
FROM  ReferencePaper X
WHERE  X.PublicationDate > 1990 AND X.language = {English}
            AND X.Topics = {correlation sea surface temperature ozone hole}
```

6.6.2.2 Medium Complexity Query

English Text: Select all stations from the CALNEVA dataset with precipitation > 6 inches; sort by date; Use the dates to look at data in Frank Gehrke's SNOW database and to derive for each date snow ratio=snow depth/precipitation.

Explanation: This query involves the joining of two separate dataset, the CALNEVE precipitation dataset and the Gehrke SNOW dataset. The joined values of snow depth and precipitation on the same date at the same station are used to calculate the snow ratio at each station on each separate data. The resulting list is then sorted by station and by date.

Structured Query Language Text:

```
SELECT (X.date, SnowRatio =  (X.snowDepth / Y.precipitation ).sortBy( X.date)
FROM SNOW X,  CALNEVA Y
WHERE   X.date = Y.date
      AND   Y.precipitation.asInches() > 6
```

6.6.2.3 Coincidence Search Query

In this query we try to demonstrate the kinds of complex coincidence searching that need to be specified by Earth scientists while conducting their inter-disciplinary research.

English Text: Find all examples of ocean blooms in Sea-WiFS ocean color data which are coincident with ozone concentrations of less than 50% for the period starting in 1990 to the present and at southern latitudes south of 55 degrees south.

Explanation: There is some evidence that toxic ocean blooms are linked to ozone depletion. In this query the GOMR data on ERS-2 will be searched to identify areas of ozone depletion (i.e the so-called ozone hole) below the defined value (this will vary temporally). Then use this dynamic location data to direct the use of a "bloom" detection algorithm in the Sea-WiFS data which is coincident with the ozone depletion.

Note that this assumes a content based search (either algorithmically or via manual inspection) has been performed on the data either at the client's site or on ingest to determine the ozone concentration contours in the GOMR data and similarly for the identification of the spatial extent of ocean blooms in the Sea-WIFS ocean color data. Content based searching by the ECS is not part of the ECS baseline. ECS will provide product data to requesting clients and any content based search and subsetting will be conducted at the client's site.

Structured Query Language Text:

```
SELECT  X.subset(OceanBlooms())
FROM  SEAWIFS X, GOMR Y
WHERE
   INTERSECT   USING  spatialOutline, timeStamp
       [   X.spatialOutline (Degrees, LatitudeSouth) < -55 AND
           X.timeStamp  >= 1990]
    WITH
       [SELECT  Y.subset(ozoneConcentration("<", "50%")).spatialOutline()
              , Y.timeStamp
         FROM  GOMR Y
         WHERE  Y.ozoneConcentration() < 50%  AND
              Y.timeStamp  >= 1990  AND
              Y.spatialExtent (Degrees, LatitudeSouth) < -55] ]
```

6.6.3 Applicable Standards

6.6.3.1 SQL3

In 1993, the ANSI and ISO development committees decided to split future SQL development into a multi-part standard. The Parts are:

Part 1: Framework A non-technical description of how the document is structured.

Part 2: Foundation The core specification, including all of the new ADT features.

Part 3: SQL/CLI The Call Level Interface.

Part 4: SQL/PSM The stored procedures specification, including computational completeness.

Part 5: SQL/Bindings The Dynamic SQL and Embedded SQL bindings taken from SQL-92.

Part 6: SQL/XA An SQL specialization of the popular XA Interface developed by X/Open

In the USA, the entirety of SQL3 is being processed as both an ANSI Domestic ("D") project and as an ISO project. The expected time frame for completion of SQL3 is currently 1998.

The SQL/CLI and SQL/PSM are being processed as fast as possible as addendums to SQL-92. In the USA, these are being processed only as International ("I") projects. SQL/CLI should be completed by the end of 1995. SQL/PSM should be completed sometime in 1996.

In addition to the SQL3 work, a number of additional related projects include:

SQL/MM An ongoing effort to define standard multi-media packages using the SQL3

ADT capabilities.

Temporal SQL

Remote Data Access (RDA)

The SQL3 standardization effort is oscillating based on the influence of the SQL-92 community and the emerging distributed object community lead by OMG and ODMG. The current (September 1995) expectation (based on inputs from Don Deutsch <301-897-1639>, the chair of

the X3H2 committee, the standards committee that is defining SQL3) is that if all goes well, the SQL3 standard recommendation by the ISO -H2 committee will be issued in mid-1996 with standardization 18-24 months later in lat 1997 or early 1998. There are sub-committees currently addressing the incompatibilities between SQL3 and the OMG query services standardization (lead largely by the ODMG group). A third influence on the de-facto standards is the involvement by Microsoft in the OLE2 specification. Since Microsoft has a large installed base most of the major database management vendors are tailoring their products to work in that environment as well.

SQL3 is a data access language, not a client/server protocol. The RDA protocol, which is a client/server protocol, could carry SQL3 but has NOT been defined for SQL3 yet since SQL3 is not yet a standard. The RDA protocol has only been defined for entry level SQL, (the SQL-89), but is expected to be expanded to support SQL3. Major competition for the RDA extension is the distributed object framework work lead by Microsoft via the OLE2 work and by OMG and ODMG.

The most compatible standard for the Release B effort given the design guidance of OMG CORBA compatibility is the use of ODMG's OQL standard. Neither DBMS products currently set as the baseline for ECS (Sybase and Illustra) have plans to follow the ODMG OQL standard. The most effective solution would be to follow the joint task force effort on compatibility between SQL3 and ODMG OQL and identify the intersection that satisfies the ECS requirements.

6.6.3.2 ODMG ODL / OQL

The ODMG-93 specification includes an introductory chapter explaining the goals and discussing architectural issues, how the standards fit together; an object model chapter, which is an extension to the OMG object model; an object definition language (ODL) standard which provide a programming-language-independent mechanism to express user object models (schema) and is an extension to the OMG IDL; an object query language (OQL) which provides a declarative access interface for interactive and programmatic query as an extension of SQL; a binding to C++ for all functionality, including object definition, manipulation, and query; a similar binding for Smalltalk; an appendix mapping the object model to OMG's; an appendix mapping the architecture to OMG's ORB; an appendix suggesting enhancements to ANSI C++ which will allow better language integration and facilitate other, more general application needs in C++.

The ODMG effort might be considered analogous to early work on the SQL standard for relational systems. However, note that ODMG had no de-facto language to start with, and had to produce a standard that spans much more, including integration with application programming languages such as C++ and Smalltalk. Substantial creative effort has been invested in ODMG-93.

The result of the ODMG-93 release does not cover all possible areas of functionality; for example, it does not cover distributed database interoperability or versioning. However, it covers all the basic capabilities necessary for an application to use an ODBMS, to create, modify, and share objects. Applications written to these interfaces will operate across all compliant ODBMS implementations with a re-compile. Continuing work is planned for later releases that will address added functionality, will track evolving related standards (e.g., changing C++), and map to further languages and domains.

6.6.3.3 Remote Database Access

One possible protocol that could carry SQL3 in a distributed system is RDA. RDA is a standard that is specified in 2 parts - a general part and a specific part. The general part defines the generic protocol while the specific part specifies the particular data access language carried by RDA. At some point in the future, additional data access languages (standards) can be defined for RDA. Currently RDA is only defined for entry level SQL (SQL - 89). The 1992 version of SQL has not yet been integrated into RDA. RDA will not be defined for SQL3 until SQL3 becomes a standard although there could be ad hoc vendor implementations. There is not much support for RDA in the U.S. by vendors although that could change.

6.6.3.4 39.50

ANSI/NISO Z39.50 is an application layer protocol. Z39.50 defines a communications mechanism for the purpose of information retrieval, by standardizing the procedures and features for searching and retrieving information. Specifically the standard supports information retrieval in a distributed, client and server environment where a process operating as a client submits a search request (a query) to another process acting as an information server. The server performs a search on one or more databases and creates a result set of records that meet the criteria of the search request. Z39.50 separates the user interface on the client side from the information servers, search engines, and databases, provides a consistent view of information from a side variety of sources, and allow client designers the capability to integrate information from a range of databases and servers.

In use, Z39.50 implementations are layered directly over TCP/IP or over the OSI layers. It is NOT a transport-layer protocol, competing with TCP or UDP. There is one known implementation that works directly over OSI. Although no one has implemented Z39.50 in the DCE or CORBA environment, conceptually it should work because it is an application layer protocol. There is a consortium of universities that has formed to implement object oriented Z39.50 clients and servers.

The ZIG (Z39.50 Implementors Working Group) is the group that defines the protocol standard. It is made up of implementors from various vendors, university groups, and freeware groups. Currently, the ZIG is looking into interoperability questions that arises from servers implementing different attribute sets. Different servers support different attribute sets and combinations of attributes. The server will let the client know when it doesn't support a particular attribute.

The ZIG is working towards making interoperability between different implementations easier to achieve. However, different profiles of Z39.50 are not guaranteed to be interoperable. Great care has to be taken to define an earth science protocol that remains interoperable with other Z39.50 implementations.

There has been progress made in the interoperability of Z39.50. A Z39.50 implementation can have extensions that other Z39.50 Servers do not support yet be interoperable for the shared common functionality. There is work being done by the ZIG such that if queries on unsupported attribute sets come in, the Server will ignore the query it cannot understand. Also, the EXPLAIN function (downloading of supported attribute sets from Server to Client on client request) should help the interoperability issue. A Server will download the names of the attributes it supports as well as a definition of the attribute (for the user) in response to an EXPLAIN request. The EXPLAIN functionality will be part of Version 3.

If there is a core set of attributes defined and then smaller, varying sets of attribute extensions, through the use of EXPLAIN, an interoperable system can be defined. An attribute set is not technically part of the Z39.50 standard. Different attribute sets can be defined. Publicizing the attribute sets will integrate the attribute set into use by other systems. There is expected to be some work done in the future that maps attributes to logical or conceptual attributes so that attributes called by different names but having similar characteristics could be mapped. However, this is not yet clearly defined.

There are working implementations of Z39.50 protocol being run over SQL databases. The Z39.50 protocol can be translated to SQL queries. The only SQL functionality that The X3H2 committee will also be defining the multi media data type (vectors, sequences, text, spatial, images, photographs, motion pictures) for use with SQL. It is not close to being defined in a definitive way yet. The committee plans on stabilizing the user defined data type in SQL3 which will make possible arrays, lists, bags, blobs, etc. Date/time data types and bit strings data types are also being considered.

Unstructured data is particularly suited to the Z39.50 application. A new data type, GRS (Generalized Record Syntax), is part of the Version 3 proposed standard. The GRS acts as a container or tree where any data type can be put in the leaf nodes of the tree so that very complex data types that are mixtures of real, integer, text, images, real-time video, etc. can be defined at the Server. A browse image together with a text legend and a brief text description of the image, can comprise a GRS data type. Fielded searches can be done on the description of the image as well as the legend field which can result in the retrieval of the image plus the accompanying text fields.

The segmentation introduced at the application level in Z39.50, for Version 3, allows the application to begin to process a segment before the entire structure is transported. This is particularly helpful when dealing with very large data such as that to be used in ECS.

6.6.3.5 HyperText Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) has been in use by the World-Wide Web since 1990 and specified by the Internet Engineering Task Force (IETF). IETF is the protocol engineering and development arm of the Internet and is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

HTTP is a standard application-level protocol with the simplicity and speed necessary for distributed, collaborative, hyper media information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands). A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

There are several efforts ongoing that are building interoperability between OSF DCE and OMG CORBA. The efforts will allow WWW clients to use the HTTP and HTML access to WWW server gateways translating into the DCE or CORBA protocols. Results from the native DCE and CORBA servers are formatted as HTML documents and passed back to the requesting client application or browser.

HTTP will interoperate with Z39.50 through Z39.50 URLs that can be put into text documents. When the user browses through the text documents with the Mosaic browser, the clicking on of the Z39.50 URL will cause that URL to be sent to a Z39.50 helper application called by the WWW Browser. The Z39.50 helper application will process the Z-URL, create a Z-Association (protocol session) with the referenced Z39.50 Server, pass along the search message to the Z39.50 Server, receive the results from the Server, and display the results to the user.

When the Z39.50 returns a set of objects, they may be documents themselves which are viewable by the original browser. Currently there is no predefined decision on whether the returned document should be viewed by the original browser or if an another helper application should be spawned. The original browser should be able to read the returned document. However, that is an application/implementation decision.

6.6.3.6 Relationships Among Information Retrieval Standards and ECS Release B Baseline Design

The relationships among the standards and protocols involve the protocols used at both the communications and applications layers, the query language or structure to be used, and the data models used by each standard. The implications on the Release B design for each of the options are that a gateway must be design and built that would translate HTTP, SQL/SQL3 protocols, and Z39.50 protocols into the Distributed Object Framework within ECS. This hybrid view of ECS supports a federated system concept for both data and service access and retrieval. ECS client toolkits will provide the API's and the data format specifications for user's to build their own client applications with full access to ECS services. The gateway options provide interoperability with existing and evolving standards in the information retrieval community.

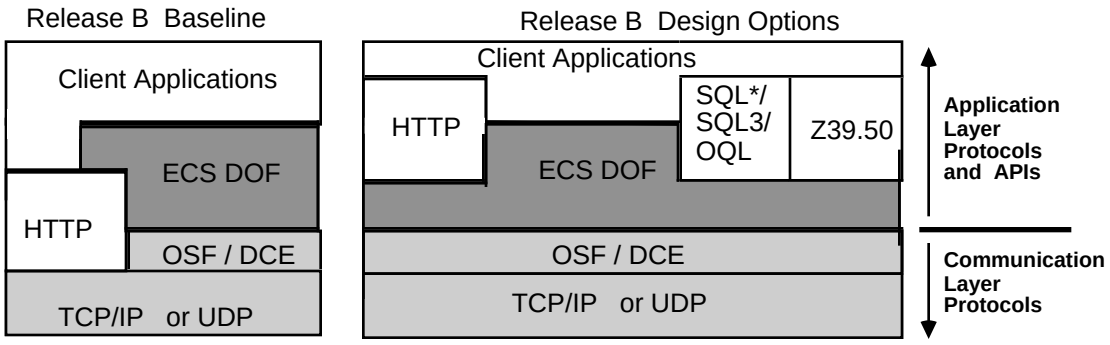

Figure 6.6-1. Release B Protocol Stack Options

The relationships among the various search specification options (query services) relate to six interrelated efforts:

- Existing vendor specific extensions to the SQL-92 standard that address ECS specific requirements for spatial, temporal, and geophysical data type searching and method invocation, such as those used in Sybase and Illustra DBMS products
- X3H2 SQL3 Standards related to SQL extension supporting an object oriented model for information access based on data type definitions

- ODMG/OQL Standards related to the OMG Object Query Services specification supporting queries on distributed objects
- Z39.50 Standards related to five types of query specifications within the Z39.50 protocol specification
- WWW HTTP and HTML standards related to extensions to the hypertext access addressing hyper-media access and the use of the query facility in the URL specification.
- The ECS Release B design of a Query object specification that will accommodate the encapsulation of parameter-value lists, search constraints, and the optional inclusion of a query language specification.

The data model aspect of the standards focuses on the differences between the X3H2 SQL3 data model assumptions and those of the OMG and the ODMG. The ECS data model is specified as a set of Earth Science Data Types with strongly typed attributes and operations. The OMG and ODMG model is accommodated by the IDL specification of the Distributed Object Framework and the SQL3 model is accommodated by the use of the Data Dictionary specification of the Earth Science Data Types.

6.6.4 ECS Release B Design Considerations

The Release A design presented an interface to the external clients via the V0 gateway. This design presented an ECS view as one of the V0 service providers and allowed users to use the V0 system IMS user interface and data model to access a subset of the planned ECS services and data collections. The options, illustrated in Figure 6.6-2, available to the Release B designers are:

- Use the V0 gateway model and provide gateway access to several protocols and languages which are translated into the ECS DOF ESDTReferenceCollector and Query object abstractions
- Design a gateway for WWW HTTP access with query extension to the URL to include query specifications as either SQL, SQL3, OQL, or PVL expressions.
- Design a gateway for ECS extensions to SQL that will evolve to an SQL3 gateway as the SQL3 standard matures and products become available in the 1997 time frame.
- Design a gateway for Z39.50 access with Earth Science specific profile extensions
- Design an ECS specific Distributed Object Framework access standard based on the internal ESDT data model, ESDTReferenceCollector and Query object, and work within the OMG standardization efforts for CORBA Facilities as an Earth Science Specific facility.

In the WWW HTTP gateway the URL would be extended to take advantage of Earth Science specific search attributes and service access as either a new Earth Science scheme or as an extension of the "?" query facility within the URL description. Either an ESQL string or a search data structure would be passed with the URL.

In the SQL* (evolving to an SQL3 or OQL standard) gateway, if the Remote Database Access protocol evolves to support SQL3, use RDA with SQL* strings passed as the search specification.

In the Z39.50 gateway the Z39.50 protocol is used and the Type-N queries are passed as the search specification.

In the ECS DOF access mode the ECS client could maintain a proxy of the ESDTReference Collector as the primary interface running on top of OODCE and passing either a search data structure (capturing the parameter-value pairs, the spatial, and temporal constraints) or an ESQL (i.e. SQL*) search specification.

6.6.5 Requirements Summary

The overall requirements for data types, methods, and operations to be defined in the Earth Science Query Language are contained in the following documents:

- ECS Level III Requirements
- ECS Level IV Requirements for Release A and B
- ECS Data Model (Conceptual and Logical)
- ECS Release A Science Data Server PDR and CDR design documents
- ECS Release A Document Data Server PDR and CDR design documents
- ECS Release B Data Management IDR documents
- ECS Release B Science Data Server IDR documents
- ECS Release B Document Data Server IDR documents

The following list provides a broad overview of the ECS search requirements for distributed and site data search, for core metadata and product specific data attributes and services, and for the integration of the search results at the client's workstation. A complete list of the Level III requirements relevant to the ESQL study and a description of the impact on the ESQL are presented in referenced requirements documents.

- Distributed and Local Search Requirements
  - Multiple Data Collections (Within a Site and Across Sites)
  - Provide Location Transparency
  - Support novice, intermediate and advanced users
- Core Metadata and Product Specific Data Values
  - Named Attributes and Services (Space, Time and Parameter)

Geographic Metadata values and Geometric Shape Constraints

Temporal Constraints

  - Text Field Matching (word, phrase, substring, expression)
  - Coincident Occurrences in space, time, & geophysical parameter
- Integration of Search Results
  - Iterative Searching
  - Intermediate Heterogeneous Results

6.6.5 ECS Release B Design Impact Analysis

The Release B baseline design for the development of a structured language representation of a user search specification (i.e. an ESQL based query) is to use a subset of the Illustra SQL extension

in support of data type extended relational models. The longer range plan is to target the SQL3 standard as it evolves and matures but to focus on and influence the Illustra implementation of a subset of the SQL3 specification. The Release B baseline for querying includes basic attribute, spatial, and temporal expressions. Full coincident search capacities will be included in later releases.

Both the SQL3 standard and the Illustra SQL extended with data types provides for a wide range of boolean expressions including attribute value matching across data types and integration of data type operations with the query expression. The Release B design and prototyping teams will map the search requirements onto the capabilities provided by the Illustra query language. The resulting subset will then be documented as part of the prototyping and detailed design, implementation, integration, and test phases of Release B.

Three Release B subsystems are primarily impacted by the integration of a structured Earth Science Query Language: Client, Data Management, and Data Server. The design issues discussed in this section include:

- Client
- Look and Feel of the User Interface to include more complex boolean expressions, matching queries, and coincident search specifications
- Advanced user support of complex query specifications not expressible using the graphical user interface
- Interface between the Client and the data management and data server API's (specifically the Data Server Interface and the Query Object Definition)
- Data Management
- Parser for the ESQL based Query Object.
- Mapping of Conceptual Schema onto the Data Server external schemas
- Science Data Server
- Query Object Management via the Distributed Object Framework and/or the Illustra DBMS Type Manager
- Mapping of Conceptual Schema onto the Internal Schema within Illustra

The design issues in each of these subsystems related to query management are in large part independent of the use of an ESQL or a GLParameterList representation of the user query. The design impact centers on the creation, parsing, and interpretation of the Query Object. The Query Object should be design to encapsulate the query specification, to provide interfaces to create standard query clauses such as in the SQL Select, From, and Where clauses, and to support sub-query creation by the DIM and LIM.

The Client Subsystem is responsible for transforming user inputs (either via the GUI or via text based input) into an instance of a query object that contains i) the specification of the results to be returned (e.g. as specified in the Select clause), ii) the specification of the data types from the conceptual model (e.g. ESDT's), and iii) the boolean expression relating attributes, values, and relationships among data types. The Release B requirements drive the content of the query expression, such as the complexity of the boolean expression, the inclusion of matching type queries, and the exclusion of coincident search. The use of ESQL as the internal representation is

independent of the content of the required queries. A common query representation within the Client subsystem would be desirable but not critical. The change in representation to an ESQL syntax could occur at the interface to the search servers (such as the DIM, LIM, or Science Data Server). If a syntax directed editor approach is chosen, as illustrated in the following figure, the syntax of the chosen ESQL (Illustra SQL in Release B) must be embedded in the GUI interface drivers to provide a user friendly interface to the syntax of the query language.

GUI Client Extensions in support of ESQL

The Data Management subsystem is responsible for the maintenance of the data definitions via the data dictionary service, the federation of schema and parsing of system wide queries within the DIM, and the integration of schema and parsing of site specific queries within the LIM. The data dictionary service supports the user via the Client workbench in formulating syntactically valid queries and queries with valid attribute values and dependencies. The conceptual model as viewed by the user is maintained by the data dictionary service and thus feeds the inputs to the Select, From, and Where clauses of the query expression. Both the DIM and the LIM must parse the content of the incoming Query Object and therefore must understand the syntax of the query expression and the data types associated with the results. In addition, any sub queries created as a result of parsing the incoming query must be generated using ESQL syntax and then passed to the relevant search servers.

The Data Server subsystem is composed of the Science Data Server and the Document Data Server. The Document Data Server is designed primarily as a World Wide Web server with text based keyword and proximity search servers. Documents can be searched using simple data types for the documents, and simple keyword based expressions. The primary impact of the choice of a query language is on the Science Data Server, and specifically on the Distributed Object Framework layers in the server. The underlying database management system for access to the data types contained in the "higher" levels of the data pyramid is baseline in Release B as Illustra. Assuming a syntactically correct query expression is passed from the Client Subsystem to the Data Server Subsystem, the query expression may be passed directly through to the Illustra server with minimal impact on the Science Data Server design. The primary trade-off to be made is the balance between Illustra Data Type design and implementation and the Distributed Object Framework request management objects.

6.6.6 Conclusions and Summary

The Release B design baseline for search and access provides for a graphical and text based input of the search and access request, and a representation of the request via a query object that contains a complex search data structure and a text string for a query language based representation. The query is parsed by the data management components (DIM and LIM) and passed to the Data Server (SDS and Document Data Server) for final parsing and results formulation. The query language syntax and features are baselined to those supported by the Illustra product's extended SQL. The design team will track the emerging standards in SQL3, ODMG's OQL, and the OMG's OQL specifications for compatibility with the Illustra extended SQL baseline and will evolve to the appropriate standard.

As part of the Release B detailed design and prototyping phase and the ongoing Evaluation Package integration and test of the Client, Data Management, and Data Server subsystems, a

collection of science scenarios should be translated into the Illustra SQL syntax and used to evaluate the ongoing design, implementation, integration and test in Release B.

The impact on the ECS Release B design focused on three primary subsystems: Client, Data Management and Science Data Server. The primary impact in the Client involves the look and feel of the Client ESST and the hiding of the details of the ESQL syntax from the average user and the provision of an advanced interface for user's requiring access to the full power of the query language. The primary impact in Data Management is the parsing needs and the sub query generation using the ESQL (Illustra SQL) syntax. All other query optimization must be provided independent of the query representation. The primary impact on the Science Data Server is at the Distributed Object Framework layer. A straight pass through of the ESQL query object to the Illustra based implementation of the Catalog, Directory and Inventory ESDT's changes the current baseline interface. Passing the Query object directly through to the Illustra engine will simplify the DOF parsing of query object in the short term but will potentially add a dependency on the Illustra DBMS in the long term.

6.7 Mode Management

6.7.1 Mode Management Overview

Mode Management addresses the planning, initiation, execution, monitoring, and control of various system activities. These activities include operations, testing, and training. Each unique activity is classified as a mode. Mode Management enables the execution of multiple modes such that each mode functions without interfering with the other and each mode maintains data integrity throughout its execution. For example, testing a data server application within the same system that is supporting operational activities. The test version of the data server must not interfere/ interact with the operational version of the data server. In addition, it will only see and have access to interface components that have been specifically set up and initiated under the same test mode.

The mode management design does not limit the number of concurrently executing modes, however, performance considerations need to be addressed prior to the initiation of an additional mode. It will support multiple test and training mode instances, but due to data persistence issues there can only be one operational mode of execution at any given time. Once an application has been initiated within a given mode, it will remain in that mode for the life of the process.

The site Resource Manager will have a view of all the components supporting each mode. This view is provided through HP OpenView and can be configured to display all of the components from every mode on one window or to display the components associated with each mode in separate windows. Software components will be duplicated, and hardware resources will be isolated whenever possible to support an additional mode. However, there will be shared resources, both hardware and software, that require special consideration to enable mode management support.

Mode Management compliance within each subsystem is addressed in the subsystem's associated detailed design specification. However, a brief synopsis of the mode management design from a system perspective is provided in the paragraph 6.7.1.1 to enhance the reader's overall understanding of the MMS design. The Mode Management Service design overview is provided in section 6.7.1.2.

6.7.1.1 Mode Management within ECS Context

Mode Management, from a system perspective, consists of procedural activities and infrastructure control. The procedural aspects of mode management address mode planning, resource allocation, and system configuration activities. Infrastructure control ensures the software subsystems will recognize the different modes of execution and that data integrity and process distinction will be maintained within each mode.

6.7.1.1.1 Procedural Activities

The procedural activities that are required to support a mode of execution are as follows:

1. Obtain and read the test/training/simulation/etc. plan from the plan originator.
2. Obtain a unique mode identifier from the Resource Manager.
3. Determine the scope of the new activity. Will the test be inter-DAAC or intra-DAAC? What services/components are involved? etc.
4. Determine whether the addition of the new mode will impact the performance of any other simultaneously executing modes. If so, how can this impact be minimized.
5. Coordinate M&O personnel required for the support of the additional mode.
6. Use the Resource planning tool to identify and allocate the necessary hardware and software resources required to support the additional mode.
7. Notify the SMC of the intended plan. If the new mode involves multiple DAACs, the SMC may become involved in the planning process.
8. Configure the directory namespace (i.e. Cell Directory Structure (CDS)) for the new mode based on the mode identifier. This activity may not be required if the CDS had been previously configured using the same mode identifier.
9. Create a HPOV map in support of the new mode. (This activity may not be required if a HPOV map had been previously configured for the same mode).
10. Establish a new HPOV session and load mode specific map into HP OpenView.
11. Identify support data sets, test software, control files, and test procedures necessary for the execution of the new mode.
12. Establish directory partitions within the file system and databases based on the mode identifier.
13. Load support data sets, configuration files, control files (drivers), and test software into mode established partitions.
14. Initiate Mode Management Service from within the HP OpenView management environment.
15. Activate new Mode within system using Mode Management Service.
16. After mode completion backup mode associated output data sets
17. Deactivate mode using the MMS and return system to a preconfigured state.

6.7.1.1.2 Infrastructure Control

The System Infrastructure will ensure data integrity between modes and provide process distinction and separation where feasible. In the case of COTS, where running multiple instances of the executable may not be possible, a single application will be required to handle requests from multiple modes. These shared resources require special consideration to ensure integrity between modes. These capabilities will be provided as part of the system infrastructure which have been designed to accommodate mode management.

Infrastructure control is based on the mode identifier. The DCE CDS and all data partitioning will be based on this identifier. These entities will be preconfigured, as part of the procedural activities required to support a mode, to accept mode specific requests. Applications are hardcoded with a mode attribute variable where mode specific requests are required. The application obtains the mode identifier at startup which it will use for all subsequent mode specific interprocess communications and data I/O requests.

6.7.1.1.2.1 Data Integrity

Data integrity must be maintained between each software mode. For example, operational processes can never read from test/training data sets and test/training data can never be written to operational data sets. All data required to support an additional mode will be duplicated. It will be partitioned by using separate volumes or by a hierarchical directory structure within the same volume such that all reading/writing of data will be segregated between software modes. All data required to support a given mode must be clearly defined, segregated, and duplicated prior to the initiation of the new activity. The segregation of the data will be based on the mode identifier.

For data storage in the UNIX environment, the data will be segregated on the same (or different) disk volume(s) in a hierarchical directory structure based on the mode identifier. The applications will access the data using the mode identifier as part of the directory path.

For data storage within a DBMS, the data will be segregated using a separate database or tape group. Applications accessing the DBMS will pass the mode identifier to the DBMS interface class which will access the corresponding mode specific database.

6.7.1.1.2.2 Process Distinction and Separation

Process distinction and separation for custom developed applications is accomplished via DCE. The Process Framework (PF) will ensure mode specific process communication by registering each server application into the DCE CDS namespace within the appropriate mode hierarchy.

When a server starts up, it registers itself in the CDS directory structure via the PF. Each entry in the CDS is uniquely identified by the server name and it's UUID. If CDS is given the UUID, it will return the rest of the name that is actually registered. Part of the administration for DCE is to setup the CDS directory structure. This is where the group, location, and mode combinations will be initially set up. This way, when registration in CDS occurs, it is known where to place the name. One function of setting up a new mode will be to manually add a new path to the CDS directory structure for the given activity. All applications will then automatically register to this new path based on the mode identifier obtained at startup. The mode identifier will be passed from the Management Framework (HP OpenView) to the remote executable startup scripts as a command line argument. The startup script will set the mode as an environment variable, which is necessary

for assigning a mode specific UUID required for ACL management, and then also pass it in as a command line argument to the application's main function. Once an application has been initiated within a given mode, it remains in that mode for the life of the process. When clients do server lookup calls, they will only see and find the servers running within the mode they are executing.

Some applications like the Management Data Access (MDA), Subagent, and virtually all COTS products will be mode independent, i.e. a single instantiation of the application will support multiple modes. When mode specific interfacing or data I/O is required the mode identifier will be passed into the application. For example the MDA, which is mode independent, processes events and routes them to the management database. For mode management support it will extract the mode attribute from the event class and then use this to route the event to the mode specific management database via the DBMS interface class. Mode independent applications will register under the "shared" hierarchical directory structure within the CDS namespace. If an event is generated by a mode independent application, the MDA will copy to active management database(s) independent of mode. This will ensure the autonomy of each mode specific management database.

6.7.1.2 MSS Mode Management Service

The Mode Management Service (MMS) within MSS provides mode initiation, monitoring, and controlling capabilities. These capabilities are provided by HP OpenView with custom code extensions. The MMS is a custom developed application which will interface with HP OpenView via HP's *ovw* APIs and with the agents via HP's *ovsnmp* API's. A high level overview of this interface is presented in Figure 6.7.1. Communication to and from HP OpenView (and therefore the MMS) is via SNMP protocol. SNMP Gets are sent directly to the remote agent while SNMP Traps and Sets are routed through the local Deputy Agent. The Deputy Agent is used to encapsulate the SNMP call into a more reliable RPC call for transport to/from the remote host.

Through the use of Agents, each process can be controlled and monitored from within HP OpenView. The MMS will incorporate the mode management user interface directly into the HP OpenView GUI, providing methods to activate and deactivate a mode. In addition it provides a mode specific user interface for accessing CSS life-cycle control (suspend, resume, and shutdown). Monitoring capabilities are provided as standard functionality within HP Openview and will be enhanced to reflect mode specific status propagation of software system, subsystem, application, program, and process level entities. Hardware is mode independent so it's status will be reflected within every mode in which it is configured.

HP OpenView will support multiple modes through the use of separate HPOV sessions. A new session can be brought up on the same host or on separate hosts. Figure 6.7.2 shows a multi-session view of how HP OpenView can be configured on separate hosts to support multiple modes. Every session can load one and only one map. The map can have any number of submaps defined that will decompose the basic high level map representation. Each mode will have it's mode specific map (and associated submaps) predefined to recognize and support the hardware and software components that are supporting the given mode. Submap context will be determined based on the mode identifier.

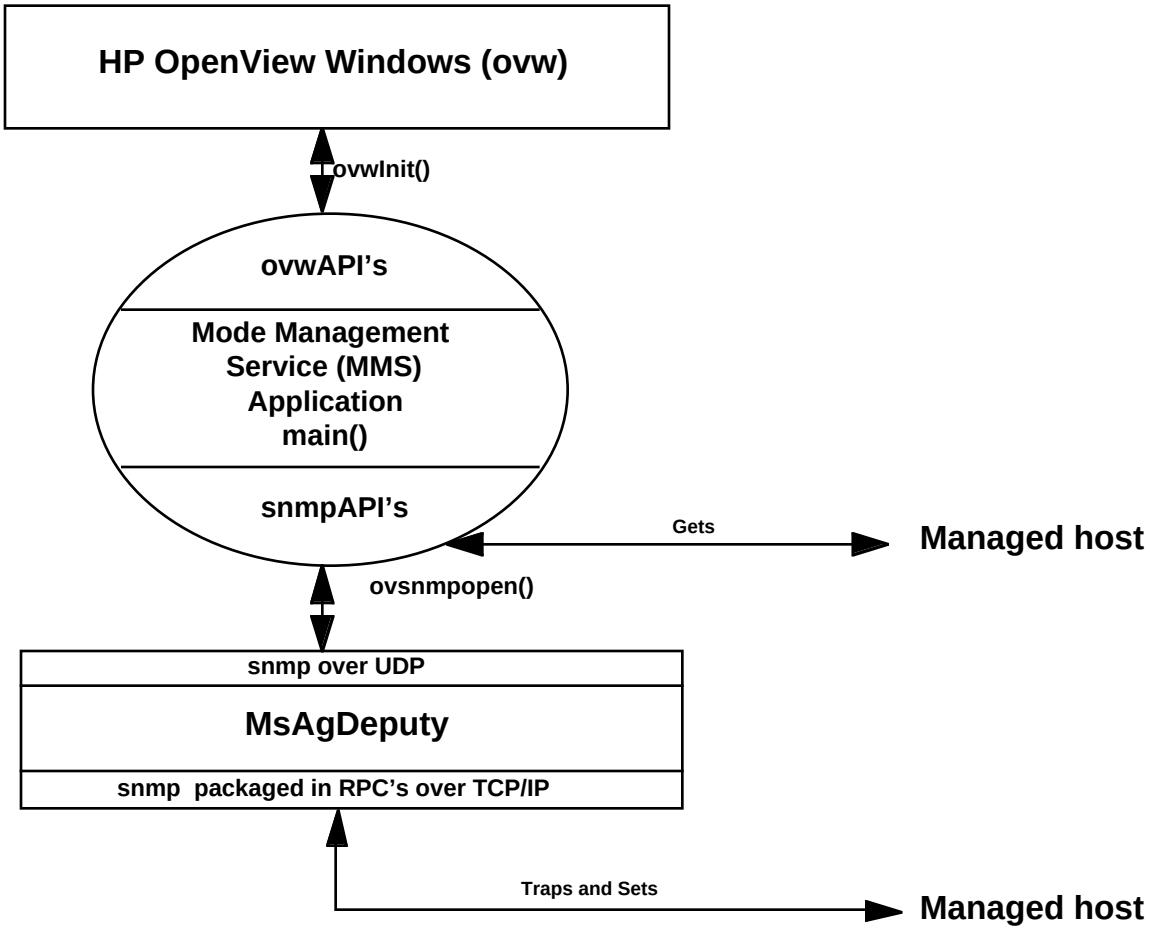

Figure 6.7-1. Mode Management Service Interface Overview Diagram

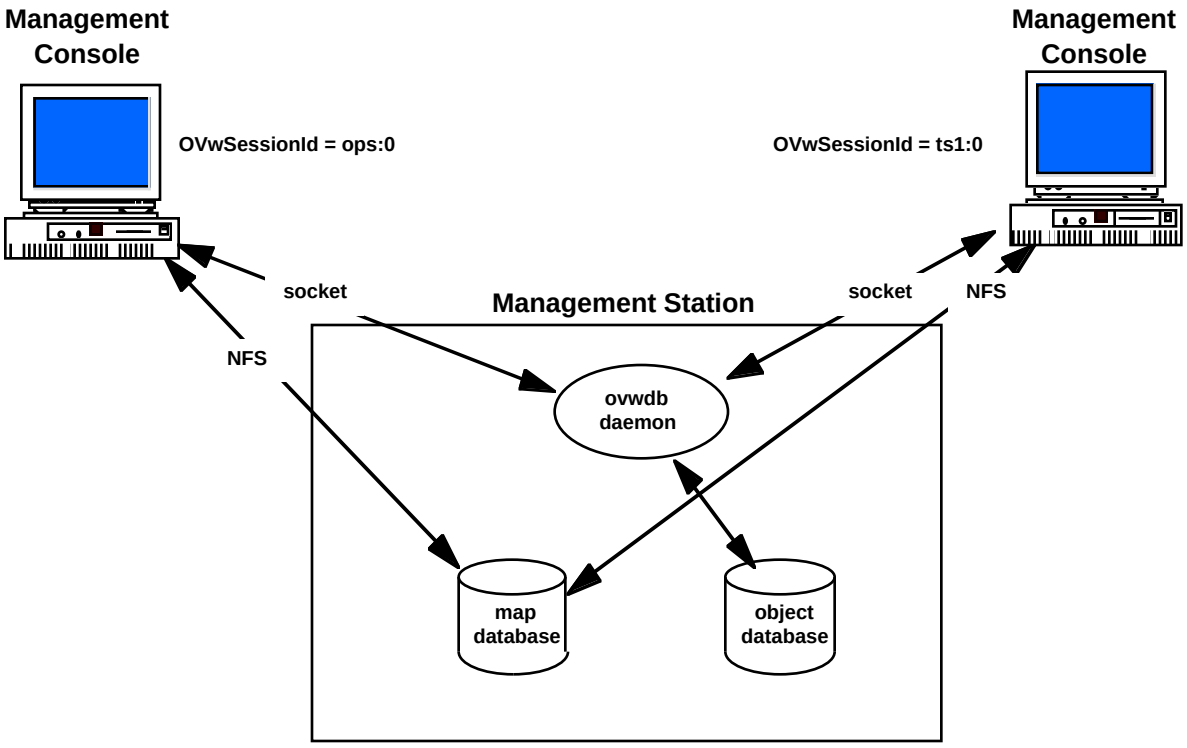

Figure 6.7-2. HP OpenView Multi-Session View Diagram

The MMS, initiated from within HP OpenView, is an event driven application. It will initialize the *ovw* and *ovsnmp* API calls, register the MMS callbacks, and then enter a main event loop. This functionality is similar to X-windows processing. Then when an action occurs, such as an operator selecting "activate mode" from the HP OpenView GUI, an event is triggered and the applicable callback is executed. The objects detailed in the MMS Object diagram are instantiated from within the callback operations.

In summary, the Mode Management Service will:

- Incorporate Mode Management Service functionality into the HP OpenView GUI.
- Support independent displays for each different mode of execution.
- Provide methods for activating and deactivating the system to a given mode.
- Enable startup/shutdown/suspend/resume activities for each process by utilizing CSS provided life cycle services.
- Provide the capability to enter a simulated time value for any none "ops" mode if required.
- Enable application/program/process level monitoring within each mode.